US006728892B1

(12) **United States Patent**
Silvkoff et al.

(10) **Patent No.: US 6,728,892 B1**
(45) **Date of Patent: Apr. 27, 2004**

(54) **METHOD FOR CONSERVING POWER IN A CAN MICROCONTROLLER AND A CAN MICROCONTROLLER THAT IMPLEMENTS THIS METHOD**

(75) Inventors: **William J. Silvkoff**, San Jose, CA (US); **Hartmut Habben**, Hamburg (DE); **Neil E. Birns**, Cupertino, CA (US)

(73) Assignee: **Koninklijke Philips Electronics N.V.**, Eindhoven (NL)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/474,901**

(22) Filed: **Dec. 30, 1999**

Related U.S. Application Data

(60) Provisional application No. 60/154,022, filed on Sep. 15, 1999.

(51) **Int. Cl.**[7] **G06F 1/26**; G06F 1/28

(52) **U.S. Cl.** **713/320**; 713/323; 713/324

(58) **Field of Search** 713/322, 320, 713/401, 300, 201, 323, 324; 710/15

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,404,544	A *	4/1995	Crayford	713/310
5,487,181	A *	1/1996	Dailey et al.	455/575
5,493,684	A *	2/1996	Gephardt et al.	713/322
5,726,541	A *	3/1998	Glenn et al.	318/16
5,848,281	A *	12/1998	Smalley et al.	713/322
5,872,903	A *	2/1999	Iwata et al.	395/551
6,021,500	A *	2/2000	Wang et al.	713/320
6,047,378	A *	4/2000	Garrett et al.	713/300
6,047,380	A *	4/2000	Nolan et al.	713/324
6,092,207	A *	7/2000	Kolinski et al.	713/323
6,279,048	B1 *	8/2001	Fadavi-Ardekani et al.	710/15
6,311,276	B1 *	10/2001	Connery et al.	713/201

(List continued on next page.)

OTHER PUBLICATIONS

H8/532 Hardware Manual, Jan. 1992. pp. 31–34, 245–269, 307–313.*

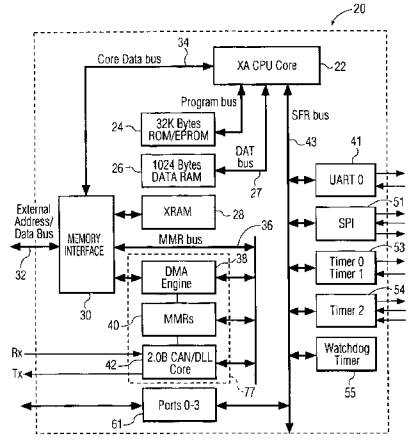
"XA–C3:Xa–16–bit microcontroller family", Philips Data Sheet,, Jan. 25, 2000, XP002165022, pp. 1–60.

Primary Examiner—Thomas Lee
Assistant Examiner—Chun Cao
(74) *Attorney, Agent, or Firm*—Aaron Waxler

(57) **ABSTRACT**

A method for conserving power in a CAN microcontroller that includes a processor core and a CAN/CAL module that includes a plurality of sub-blocks that cooperatively function to process incoming CAL/CAN messages, which method includes the steps of placing the processor core in a power-reduction mode of operation (e.g., a sleep or idle mode of operation), placing the CAN/CAL module in a power-reduction mode of operation, and activating the CAN/CAL module to process an incoming CAL/CAN message (e.g., to perform automatic hardware assembly of a multi-frame, fragmented CAL/CAN message), thereby terminating the power-reduction mode of operation thereof, while the processor core is in its power-reduction mode of operation. In a preferred embodiment, the CAN/CAL module automatically assembles incoming, multi-frame, fragmented messages while the processor core remains in its power-reduction mode of operation, and the CAN/CAL module generates a message-complete interrupt in response to completion of assembly of the multi-frame, fragmented message, whereby the terminating step is executed in response to the message-complete interrupt. In another embodiment, the method includes the steps of placing the entire CAN microcontroller, including both the processor core and the CAN/CAL module in a power-down mode of operation, detecting receipt of an incoming message, and activating the CAN/CAL module in response to the detecting step to process the incoming message (e.g., to perform automatic hardware assembly of a multi-frame, fragmented CAL/CAN message), thereby terminating the power-down mode of operation of the CAN/CAL module, without terminating the power-down mode of operation of the processor core.

44 Claims, 8 Drawing Sheets



U.S. PATENT DOCUMENTS

6,324,597 B2 * 11/2001 Collier 710/22
6,370,599 B1 * 4/2002 Anand et al. 710/15
6,457,082 B1 * 9/2002 Zhang et al. 710/260
6,463,542 B1 * 10/2002 Yu et al. 713/320

* cited by examiner

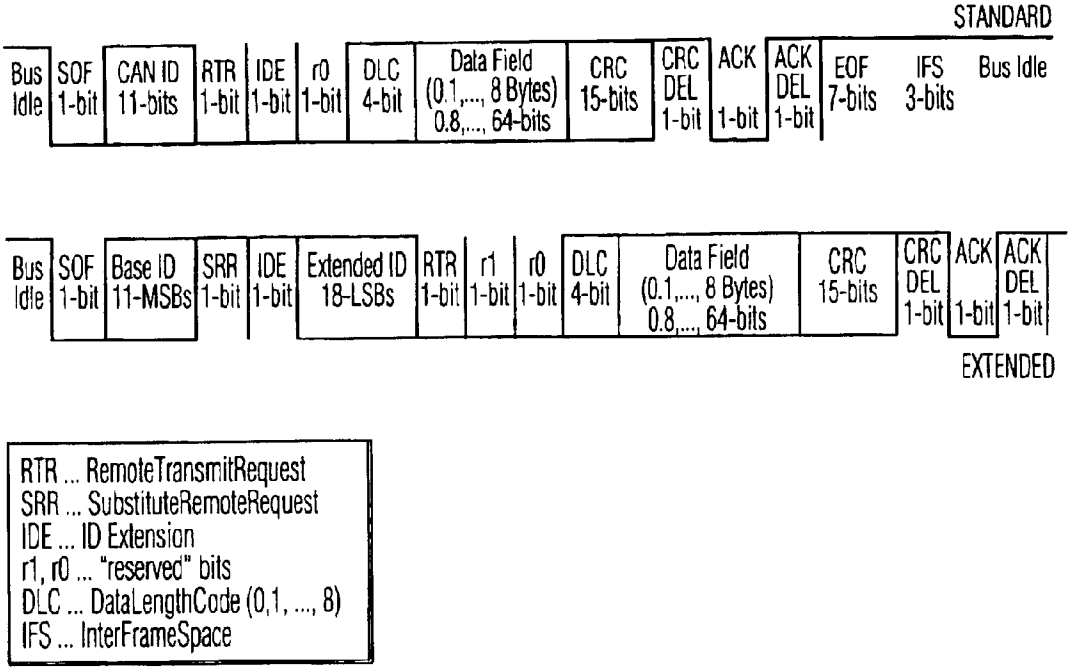

STANDARD
Bus Idle
SOF 1-bit
CAN ID 11-bits
RTR 1-bit
IDE 1-bit
r0 1-bit
DLC 4-bit
Data Field (0.1,..., 8 Bytes) 0.8,..., 64-bits
CRC 15-bits
CRC DEL 1-bit
ACK 1-bit
ACK DEL 1-bit
EOF 7-bits
IFS 3-bits
Bus Idle
Bus Idle
SOF 1-bit
Base ID 11-MSBs
SRR 1-bit
IDE 1-bit
Extended ID 18-LSBs
RTR 1-bit
r1 1-bit
r0 1-bit
DLC 4-bit
Data Field (0.1,..., 8 Bytes) 0.8,..., 64-bits
CRC 15-bits
CRC DEL 1-bit
ACK 1-bit
ACK DEL 1-bit
EXTENDED
RTR ... RemoteTransmitRequest
SRR ... SubstituteRemoteRequest
IDE ... ID Extension
r1, r0 ... "reserved" bits
DLC ... DataLengthCode (0,1, ..., 8)
IFS ... InterFrameSpace

FIG. 1

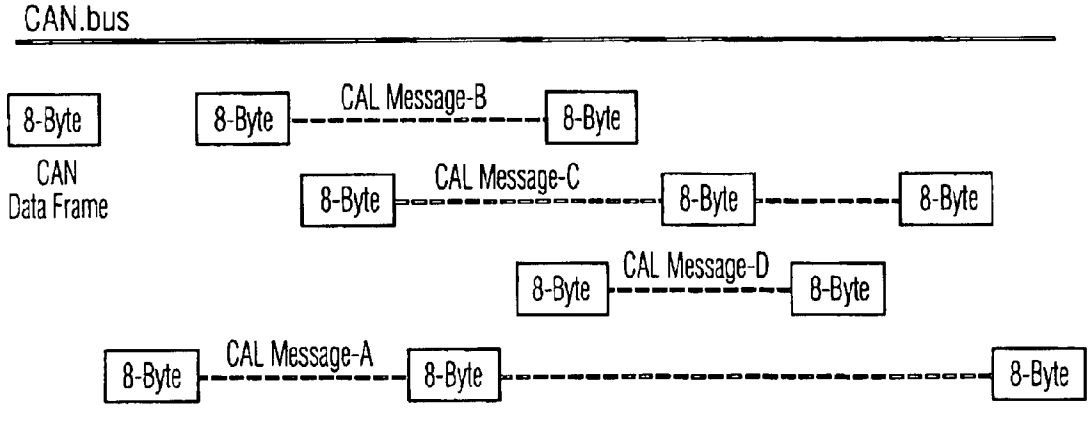

CAN.bus
8-Byte
CAN Data Frame
8-Byte
CAL Message-B
8-Byte
8-Byte
CAL Message-C
8-Byte
8-Byte
8-Byte
8-Byte
CAL Message-D
8-Byte
8-Byte
CAL Message-A
8-Byte
8-Byte

FIG. 2

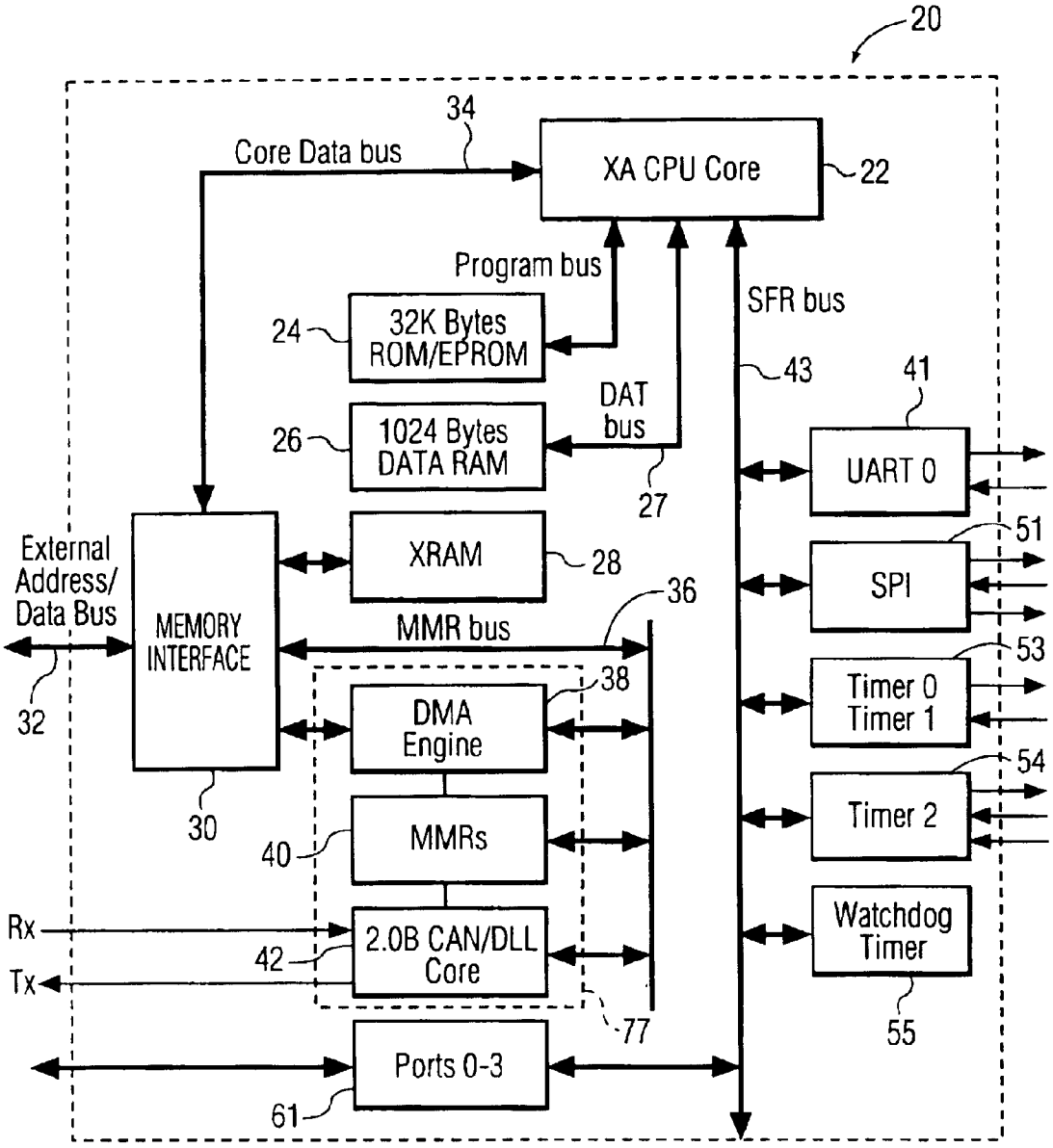

20
34
Core Data bus
XA CPU Core
22
Program bus
SFR bus
24
32K Bytes ROM/EPROM
43
41
DAT bus
26
1024 Bytes DATA RAM
27
UART 0
External Address/ Data Bus
MEMORY INTERFACE
XRAM
28
51
SPI
36
MMR bus
53
Timer 0 Timer 1
38
DMA Engine
32
30
54
Timer 2
40
MMRs
Rx
2.0B CAN/DLL Core
Watchdog Timer
42
Tx
77
55
Ports 0-3
61

FIG. 3

MMRs

MMR name	R/W?	Reset	Access	Address Offset	Description
Message Object Registers (n = 0 - 31)					
MnMIDH	R/W	x....x00b	Word only	000n4n3n2n1n00000b (n0h)	Message n Match ID High
MnMIDL	R/W	xxxxh	Word only	000n4n3n2n1n00010b (n2h)	Message n Match ID Low
MnMSKH	R/W	x....x000b	Word only	000n4n3n2n1n00100b (n4h)	Message n Mask High
MnMSKL	R/W	xxxxh	Word only	000n4n3n2n1n00110b (n6h)	Message n Mask Low
MnCTL	R/W	00000xxxb	Byte/Word	000n4n3n2n1n01000b (n8h)	Message n Control
MnBLR	R/W	xxxxh	Word only	000n4n3n2n1n01010b (nAh)	Message n Buffer Location
MnBSZ	R/W	00000xxxb	Byte/Word	000n4n3n2n1n01100b (nCh)	Message n Buffer Size
MnFCR	R/W	00xxxxxxb	Byte/Word	000n4n3n2n1n01110b (nEh)	Message n Fragmentation Count
CIC Registers					
MCPLL	R/C	0000h	Byte/Word	224h	Message Complete Low
MCPLH	R/C	0000h	Byte/Word	226h	Message Complete High
CANINTFLG	R/C	0000h	Byte/Word	228h	CAN Interrupt Flag Register
MCIR	RO	0000h	Byte/Word	229h	Message Complete Info Reg.
MEIR	RO	0000h	Byte/Word	22Ah	Message Error Info Register
FESTR	R/C	0000h	Byte/Word	22Ch	Frame Error Status Register
FEENR	R/W	0000h	Byte/Word	22Eh	Frame Error Enable Register
SCP/SPI Registers					
SPICFG	R/W	0000h	Byte/Word	260h	SCP/SPI Configuration
SPIDATA	R/W	00h	Byte/Word	262h	SCP/SPI Data
SPICS	R/W	00h	Byte/Word	263h	SCP/SPI Control and Status
CCB Registers					
CANCMR	R/W	01h	Byte/Word	270h	CAN Command Register
CANSTR	R/O	00h	Byte/Word	271h	CAN Status Register
CANBTR	R/W	00h	Byte/Word	272h	CAN Bus Timing Reg. (low)
-	R/W	00h	Byte/Word	273h	CAN Bus Timing Reg. (high)
TXERC	R/W*	00h	Byte/Word	274h	Tx Error Counter
RXERC	R/W*	00h	Byte/Word	275h	Rx Error Counter
EWLR	R/W	96h	Byte/Word	276h	Error Warning Limit Register
ECCR	RO	0000h	Byte/Word	278h	Error Code Capture Register
ALCR	RO	0000h	Byte/Word	27Ah	Arbitration Lost Capture Reg.
RTXDTM	WO	0000h	Byte/Word	27Ch	RTX Data Test Mode
GCTL	R/W	0000h	Byte/Word	27Eh	Global Control Byte
MIF Registers					
XRAMB	R/W	FEh	Byte/Word	290h	XRAM Base Address
MBXSR	R/W	FFh	Byte/Word	291h	Msg. Buff./XRAM Seg. Reg.
MIFBTRL	R/W	EFh	Byte/Word	292h	MIF Bus Timing Reg. Low
MIFBTRH	R/W	FFh	Byte/Word	293h	MIF Bus Timing Reg. High

Legend: R/W = Read & Write, RO = Read Only, WO = Write Only, R/C = Read & Clear, W* = Writable only during CAN Reset mode, x = undefined after reset.

FIG. 4

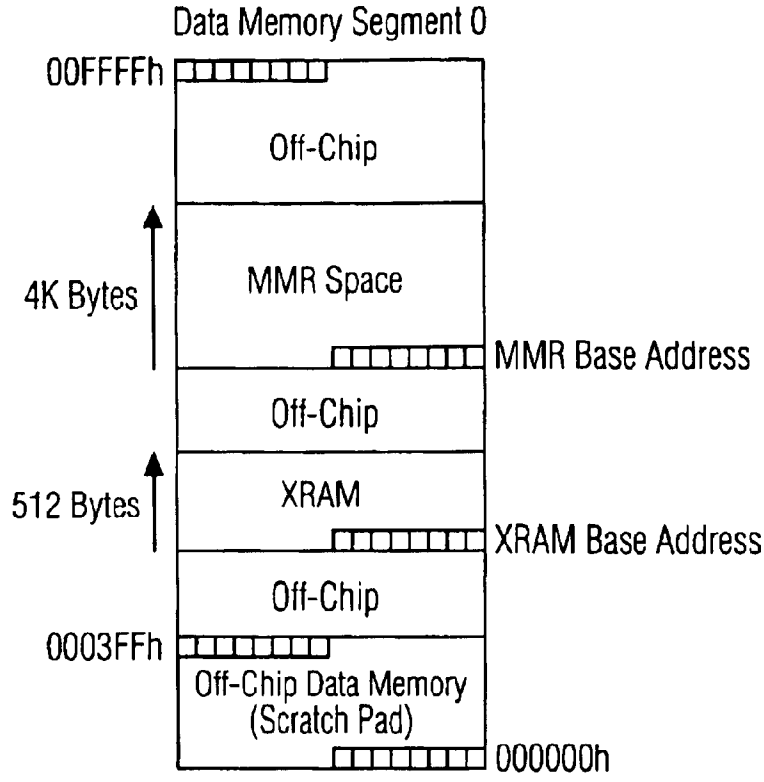

Data Memory Segment 0
00FFFFh
Off-Chip
4K Bytes
MMR Space
MMR Base Address
Off-Chip
512 Bytes
XRAM
XRAM Base Address
Off-Chip
0003FFh
Off-Chip Data Memory
(Scratch Pad)
000000h

FIG. 5

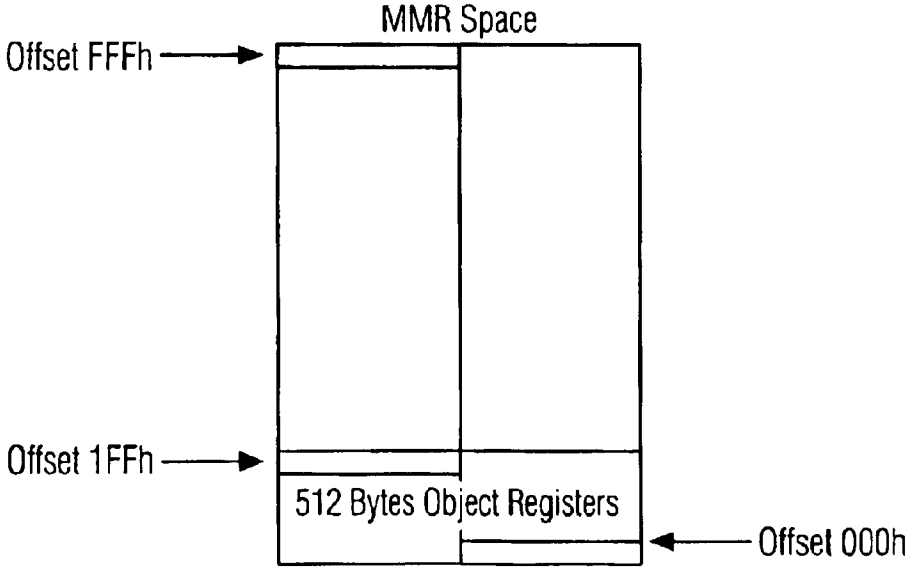

MMR Space
Offset FFFh
Offset 1FFh
512 Bytes Object Registers
Offset 000h

FIG. 6

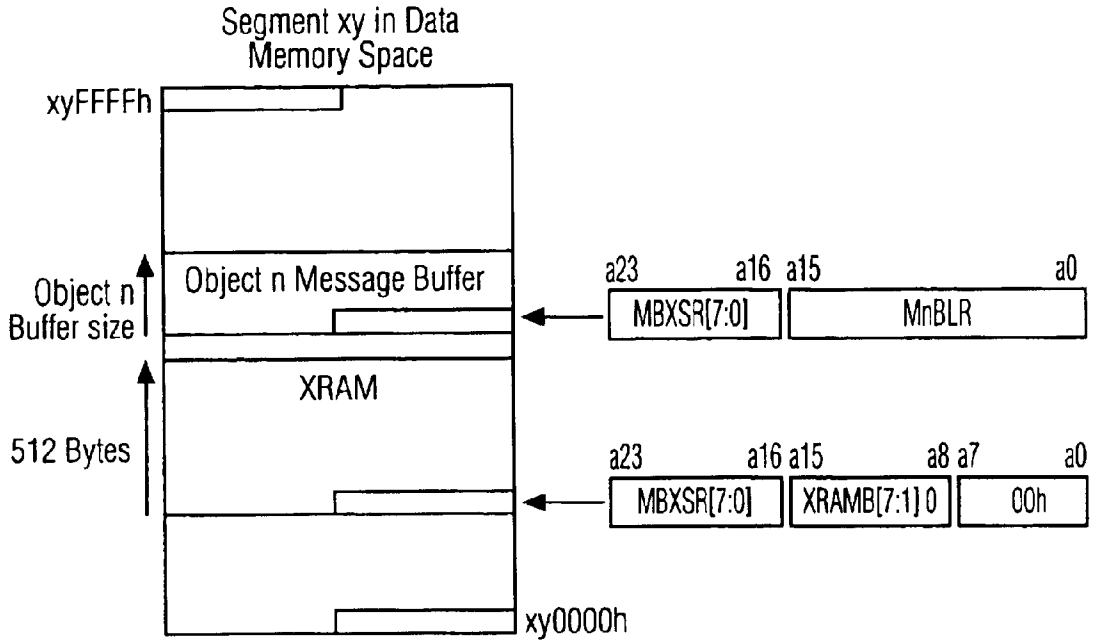

Segment xy in Data Memory Space
xyFFFFh
Object n Buffer size
Object n Message Buffer
XRAM
512 Bytes
xy0000h
a23
a16
a15
a0
MBXSR[7:0]
MnBLR
a23
a16
a15
a8
a7
a0
MBXSR[7:0]
XRAMB[7:1] 0
00h

FIG. 7

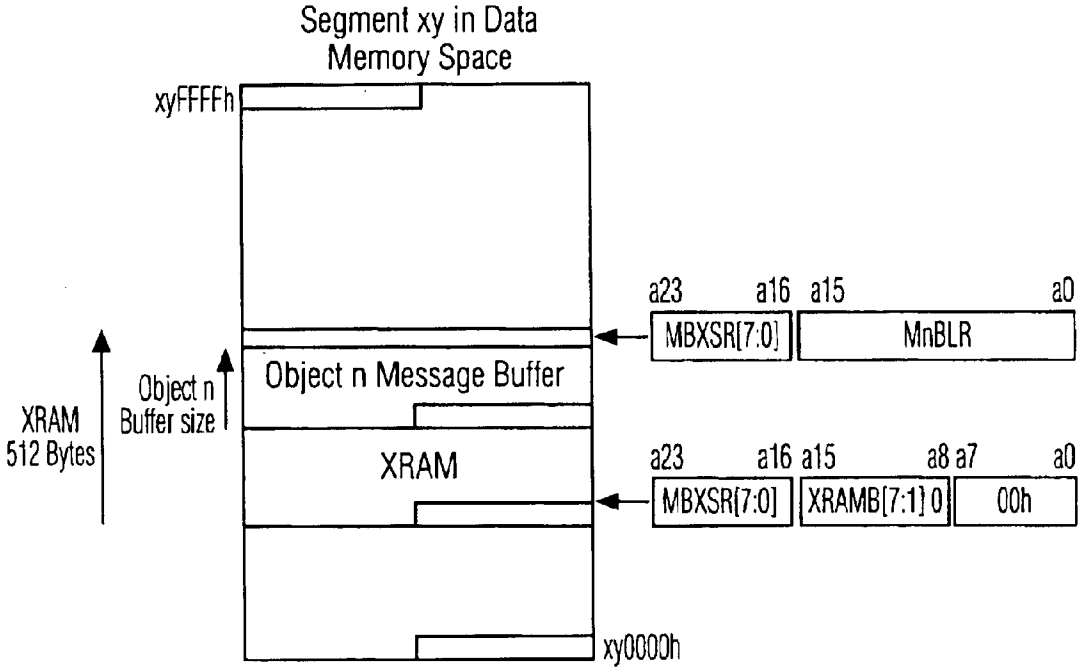

Segment xy in Data Memory Space
xyFFFFh
XRAM 512 Bytes
Object n Buffer size
Object n Message Buffer
XRAM
xy0000h
a23
a16
a15
a0
MBXSR[7:0]
MnBLR
a23
a16
a15
a8
a7
a0
MBXSR[7:0]
XRAMB[7:1] 0
00h

FIG. 8

Object n Match ID Field (MnMIDH and MnMIDL)

Mid28 – Mid18	Mid17 – Mid10	Mid9 – Mid2	Mid1	Mid0	MIDE

Object n Mask Field (MnMSKH and MnMSKL)

Msk28 – Msk18	Msk17 – Msk10	Msk9 – Msk2	Msk1	Msk0

Screener ID Field (assembled from incoming bit-stream)

CAN ID.28 – CAN ID.18	Data Byte 1 [7:0]	Data Byte 2 [7:0]	x	x	IDE

FIG. 9

Object n Match ID Field (MnMIDH and MnMIDL)

Mid28 – Mid18	Mid17 – Mid10	Mid9 – Mid2	Mid1	Mid0	MIDE

Object n Mask Field (MnMSKH and MnMSKL)

Msk28 – Msk18	Msk17 – Msk10	Msk9 – Msk2	Msk1	Msk0

Screener ID Field (assembled from incoming bit-stream)

CAN ID.28 – CAN ID.0	IDE

FIG. 10

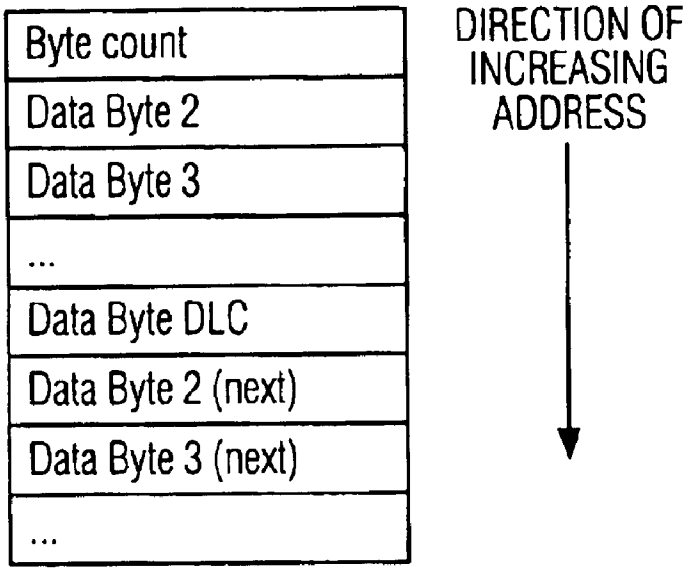

Byte count
Data Byte 2
Data Byte 3
...
Data Byte DLC
Data Byte 2 (next)
Data Byte 3 (next)
...
DIRECTION OF INCREASING ADDRESS

FIG. 11

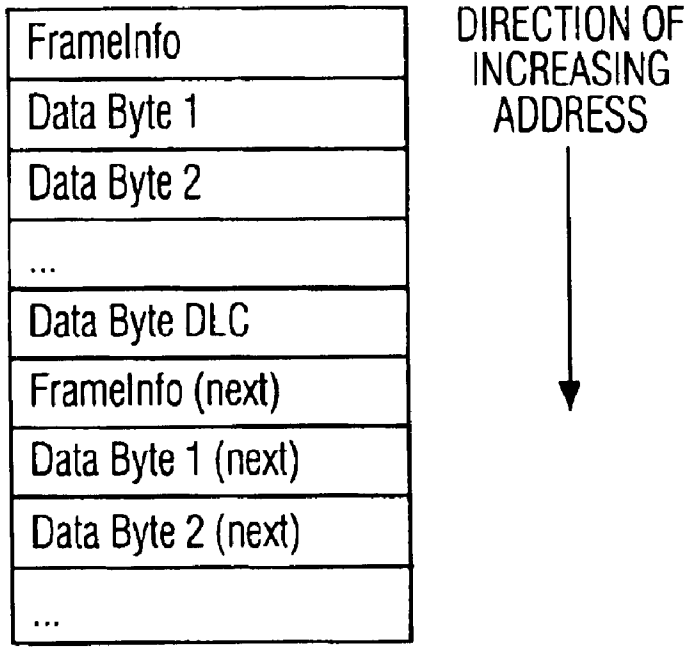

FrameInfo
Data Byte 1
Data Byte 2
...
Data Byte DLC
FrameInfo (next)
Data Byte 1 (next)
Data Byte 2 (next)
...
DIRECTION OF INCREASING ADDRESS

FIG. 12

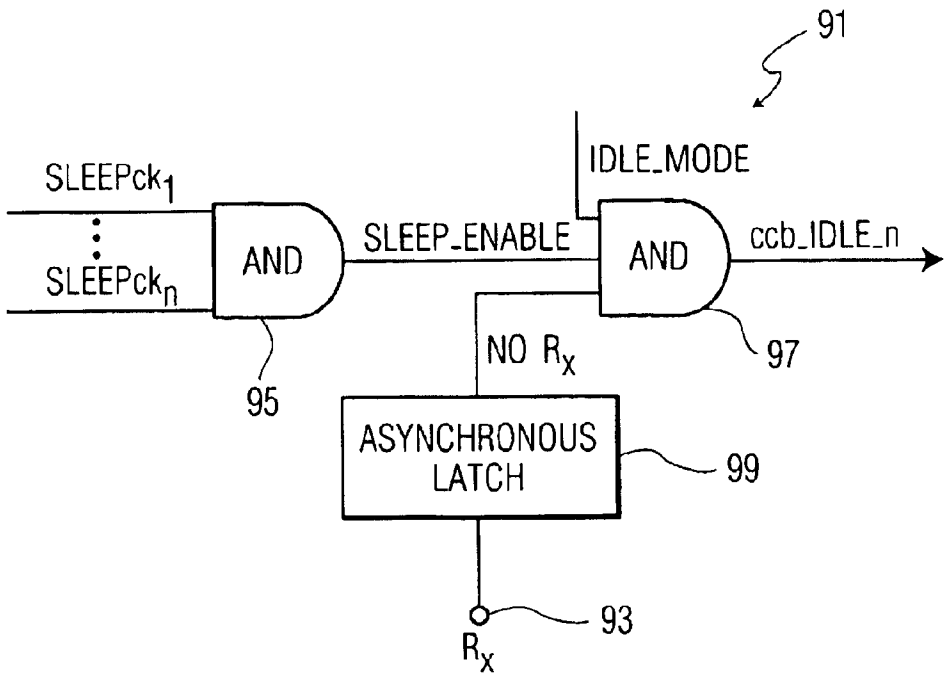

91
IDLE_MODE
SLEEPck1
SLEEPckn
AND
SLEEP_ENABLE
AND
ccb_IDLE_n
95
NO Rx
97
ASYNCHRONOUS LATCH
99
93
Rx

FIG. 13

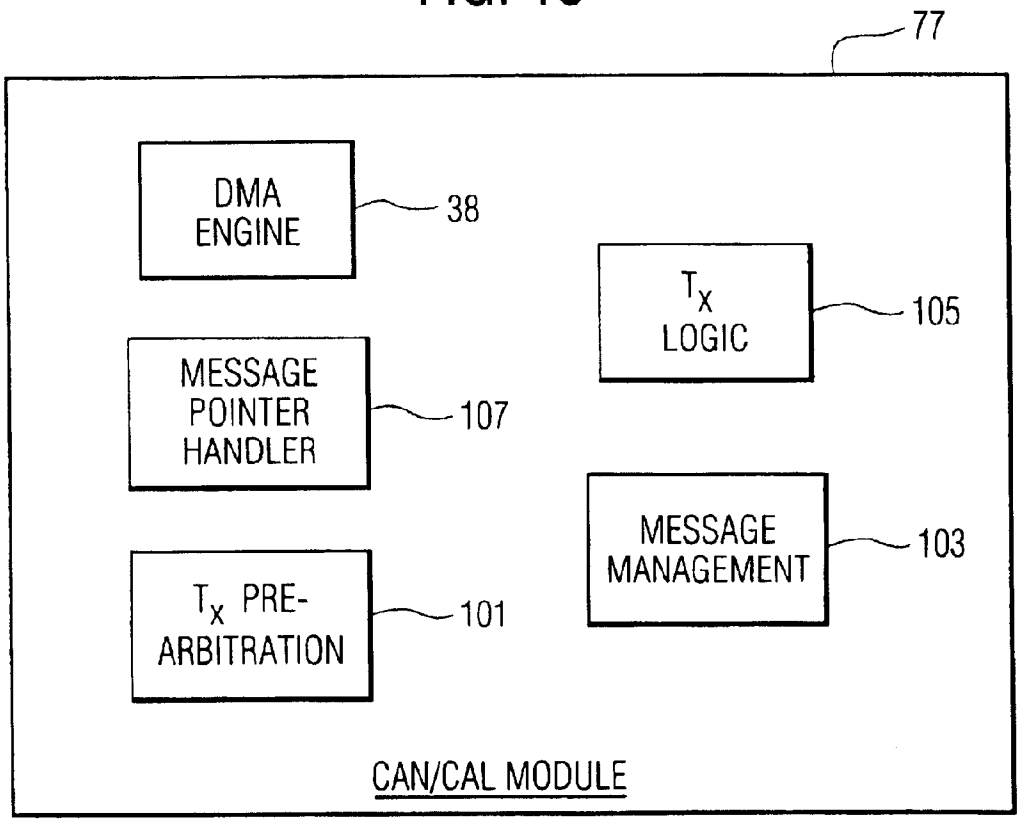

77
DMA ENGINE
38
Tx LOGIC
105
MESSAGE POINTER HANDLER
107
MESSAGE MANAGEMENT
103
Tx PRE-ARBITRATION
101
CAN/CAL MODULE

FIG. 14

METHOD FOR CONSERVING POWER IN A CAN MICROCONTROLLER AND A CAN MICROCONTROLLER THAT IMPLEMENTS THIS METHOD

This application claims the full benefit and priority of U.S. Provisional Application Serial No. 60/154,022, filed on Sep. 15, 1999, the disclosure of which is fully incorporated herein for all purposes.

BACKGROUND OF THE INVENTION

The present invention relates generally to the field of data communications, and more particularly, to the field of serial communications bus controllers and microcontrollers that incorporate the same.

CAN (Control Area Network) is an industry-standard, two-wire serial communications bus that is widely used in automotive and industrial control applications, as well as in medical devices, avionics, office automation equipment, consumer appliances, and many other products and applications. CAN controllers are currently available either as stand-alone devices adapted to interface with a microcontroller or as circuitry integrated into or modules embedded in a microcontroller chip. Since 1986, CAN users (software programmers) have developed numerous high-level CAN Application Layers (CALs) which extend the capabilities of the CAN while employing the CAN physical layer and the CAN frame format, and adhering to the CAN specification. CALs have heretofore been implemented primarily in software, with very little hardware CAL support. Consequently, CALs have heretofore required a great deal of host CPU intervention, thereby increasing the processing overhead and diminishing the performance of the host CPU.

Thus, there is a need in the art for a CAN hardware implementation of CAL functions normally implemented in software in order to offload these tasks from the host CPU to the CAN hardware, thereby enabling a great savings in host CPU processing resources and a commensurate improvement in host CPU performance. One of the most demanding and CPU resource-intensive CAL functions is message management, which entails the handling, storage, and processing of incoming CAL/CAN messages received over the CAN serial communications bus and/or outgoing CAL/CAN messages transmitted over the CAN serial communications bus. CAL protocols, such as DeviceNet, CANopen, and OSEK, deliver long messages distributed over many CAN frames, which methodology is sometimes referred to as "fragmented" or "segmented" messaging. The process of assembling such fragmented, multi-frame messages has heretofore required a great deal of host CPU intervention. In particular, CAL software running on the host CPU actively monitors and manages the buffering and processing of the message data, in order to facilitate the assembly of the message fragments or segments into complete messages.

Based on the above and foregoing, it can be appreciated that there presently exists a need in the art for a hardware implementation of CAL functions normally implemented in software in order to offload these tasks from the host CPU, thereby enabling a great savings in host CPU processing resources and a commensurate improvement in host CPU performance.

The assignee of the present invention has recently developed a new microcontroller product, designated "XA-C3", that fulfills this need in the art. The XA-C3 is the newest member of the Philips XA (eXtended Architecture) family of high performance 16-bit single-chip microcontrollers. It is believed that the XA-C3 is the first chip that features hardware CAL support.

The XA-C3 is a CMOS 16-bit CAL/CAN 2.0B microcontroller that incorporates a number of different inventions, including the present invention. These inventions include novel techniques and hardware for filtering, buffering, handling, and processing CAL/CAN messages, including the automatic assembly of multi-frame fragmented messages with minimal CPU intervention, as well as for managing the storage and retrieval of the message data, and the memory resources utilized therefor.

The present invention relates to a power conservation scheme that enables one or more hardware components of the microcontroller, e.g., the CPU core, to remain in a sleep or idle mode while other hardware components, e.g., CAL/CAN hardware components, are active, e.g., automatically assembling a multi-frame, fragmented message.

SUMMARY OF THE INVENTION

The present invention encompasses a method for conserving power in a CAN microcontroller that includes a processor core and a CAN/CAL module that includes a plurality of sub-blocks that cooperatively function to process incoming CAL/CAN messages, which method includes the steps of placing the processor core in a power-reduction mode of operation (e.g., a sleep or idle mode of operation), placing the CAN/CAL module in a power-reduction mode of operation, and activating the CAN/CAL module to process an incoming CAL/CAN message (e.g., to perform automatic hardware assembly of a multi-frame, fragmented CAL/CAN message), thereby terminating the power-reduction mode of operation thereof, while the processor core is in its power-reduction mode of operation.

In a preferred embodiment, the method further includes the steps of generating a message-complete interrupt in response to completion of assembly of the multi-frame, fragmented CAL/CAN message, and activating the processor core in response to the message-complete interrupt. In a particular preferred embodiment, the method further includes the steps of repeating the step of placing the CAN/CAL module in a power-reduction mode of operation and the activating step, in seriatim, a plurality of times, while the processor core is in its power-reduction mode of operation.

In a present specific implementation, the step of placing the CAN/CAL module in a power-reduction mode operation is performed by a power or sleep control module contained within the CAN/CAL module, in the following manner. Particularly, first logic circuitry associated with each of the plurality of sub-blocks generates a respective first signal having a first logic level if that sub-block is currently active, and having a second logic level if that sub-block is not currently active. Second logic circuitry generates a second signal having a first logic level if any of the first signals are at the first logic level, and having a second logic level in response to all of the first signals having the second logic level. Third logic circuitry generates a third signal having a first logic level if the processor core is not idle, and having a second logic level if the processor core is idle. Fourth logic circuitry generates a fourth signal having a first logic level if an incoming message is being received, and having a second logic level if an incoming message is not being received. Fifth logic circuitry generates a clock disable signal in response to the second, third, and fourth signals all being at their respective second logic level. Sixth logic

circuitry disables a clock applied to the CAN/CAL module in response to the clock disable signal to thereby place the CAN/CAL module in the power-reduction mode of operation.

In a present specific implementation, the step of placing the processor core in a power-reduction mode of operation is performed by a power or sleep control module contained within the CAN/CAL module, in the following manner. Particularly, a first logic portion of the power control module generates a clock disable signal having a first logic level if the processor core has pending interrupts, and having a second logic level if the processor core has no pending interrupts, and a second logic portion of the power control module disables a clock applied to the processor core in response to the clock disable signal, to thereby place the processor core in the power-reduction mode of operation.

The present invention, in another of its aspects, encompasses a method for conserving power in a CAN microcontroller that includes a processor core and a CAN/CAL module, which method includes the steps of placing the processor core in a power-reduction mode of operation (e.g., a sleep or idle mode of operation), while the CAN/CAL module is actively processing an incoming CAL/CAN message (e.g., to perform automatic hardware assembly of a multi-frame, fragmented CAL/CAN message), and terminating the power-reduction mode of operation in response to an interrupt.

In a preferred embodiment, the CAN/CAL module automatically assembles incoming, multi-frame, fragmented messages while the processor core remains in its power-reduction mode of operation, and the CAN/CAL module generates a message-complete interrupt in response to completion of assembly of the multi-frame, fragmented message, whereby the terminating step is executed in response to the message-complete interrupt.

In a present specific implementation, the step of placing the processor core in a power-reduction mode of operation is performed by a power or sleep control module contained within the CAN/CAL module, in the following manner. Particularly, a first logic portion of the power control module generates a clock disable signal having a first logic level if the processor core has pending interrupts, and having a second logic level if the processor core has no pending interrupts, and a second logic portion of the power control module disables a clock applied to the processor core in response to the clock disable signal, to thereby place the processor core in the power-reduction mode of operation.

In yet another of its aspects, the present invention encompasses a method for conserving power in a CAN microcontroller that includes a processor core and a CAN/CAL module, which method includes the steps of placing the entire CAN microcontroller, including both the processor core and the CAN/CAL module in a power-down mode of operation, detecting receipt of an incoming message, and activating the CAN/CAL module in response to the detecting step to process the incoming message (e.g., to perform automatic hardware assembly of a multi-frame, fragmented CAL/CAN message), thereby terminating the power-down mode of operation of the CAN/CAL module, without terminating the power-down mode of operation of the processor core.

In a preferred embodiment, the method further includes the step of placing the processor core in a power-reduction mode of operation (e.g., a sleep or idle mode of operation) in response to the detecting step. The processor core can be substantially instantaneously woken up when in the power-down mode of operation, and can be woken up over a prescribed wake-up period when in the power-down mode of operation.

In a specific implementation, the placing step is performed by determining whether the CAN/CAL module is ready to be placed into the power-down mode of operation, and stopping a main system clock in response to a determination that the CAN/CAL module is ready to be placed into the power-down mode of operation. Further, in the specific implementation, the method further includes the step of terminating the power-down mode of operation of the entire CAN microcontroller, including both the processor core and the CAN/CAL module, in response to an external interrupt or a system reset command.

In yet other aspects, the present invention encompasses a CAN microcontroller that implements any one or more of the above-discussed methods.

BRIEF DESCRIPTION OF THE DRAWINGS

These and various other aspects, features, and advantages of the present invention will be readily understood with reference to the following detailed description of the invention read in conjunction with accompanying drawings, in which:

FIG. **1** is a diagram illustrating the format of a Standard CAN Frame and the format of an Extended CAN Frame;

FIG. **2** is a diagram illustrating the interleaving of CAN Data Frames of different, unrelated messages;

FIG. **3** is a high-level, functional block diagram of the XA-C3 microcontroller;

FIG. **4** is a table listing all of the Memory Mapped Registers (MMRs) provided by the XA-C3 microcontroller;

FIG. **5** is a illustrating the mapping of the overall data memory space of the XA-C3 microcontroller;

FIG. **6** is a diagram illustrating the MMR space contained within the overall data memory space of the XA-C3 microcontroller;

FIG. **7** is a diagram illustrating formation of the base address of the on-chip XRAM of the XA-C3 microcontroller, with an object n message buffer mapped into off-chip data memory;

FIG. **8** is a diagram illustrating formation of the base address of the on-chip XRAM of the XA-C3 microcontroller, with an object n message buffer mapped into the on-chip XRAM;

FIG. **9** is a diagram illustrating the Screener ID Field for a Standard CAN Frame;

FIG. **10** is a diagram illustrating the Screener ID Field for an Extended CAN Frame;

FIG. **11** is a diagram illustrating the message storage format for fragmented CAL messages;

FIG. **12** is a diagram illustrating the message storage format for fragmented CAN messages;

FIG. **13** is a partial schematic, partial functional block diagram of a sleep control module incorporated within the XA-C3 microcontroller in accordance with the present invention; and,

FIG. **14** is a high-level block diagram depicting the CAN/CAL module of the XA-C3 microcontroller, and its constituent sub-blocks.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention is described below in the context of a particular implementation thereof, i.e., in the context of the

XA-C3 microcontroller manufactured by Philips Semiconductors. Of course, it should be clearly understood that the present invention is not limited to this particular implementation, as any one or more of the various aspects and features of the present invention disclosed herein can be utilized either individually or any combination thereof, and in any desired application, e.g., in a stand-alone CAN controller device or as part of any other microcontroller or system.

The following terms used herein in the context of describing the preferred embodiment of the present invention (i.e., the XA-C3 microcontroller) are defined as follows:

Standard CAN Frame: The format of a Standard CAN Frame is depicted in FIG. **1**.

Extended CAN Frame: The format of an Extended CAN Frame is also depicted in FIG. **1**.

Acceptance Filtering: The process a CAN device implements in order to determine if a CAN frame should be accepted or ignored and, if accepted, to store that frame in a pre-assigned Message Object.

Message Object: A Receive RAM buffer of pre-specified size (up to 256 bytes for CAL messages) and associated with a particular Acceptance Filter or, a Transmit RAM buffer which the User preloads with all necessary data to transmit a complete CAN Data Frame. A Message Object can be considered to be a communication channel over which a complete message, or a succession of messages, can be transmitted.

CAN Arbitration ID: An 11-bit (Standard CAN 2.0Frame) or 29-bit (Extended CAN 2.0B Frame) identifier field placed in the CAN Frame Header. This ID field is used to arbitrate Frame access to the CAN bus. Also used in Acceptance Filtering for CAN Frame reception and Transmit Pre-Arbitration.

Screener ID: A 30-bit field extracted from the incoming message which is then used in Acceptance Filtering. The Screener ID includes the CAN Arbitration ID and the IDE bit, and can include up to 2 Data Bytes. These 30 extracted bits are the information qualified by Acceptance Filtering.

Match ID: A 30-bit field pre-specified by the user to which the incoming Screener ID is compared. Individual Match IDs for each of 32 Message Objects are programmed by the user into designated Memory Mapped Registers (MMRs).

Mask: A 29-bit field pre-specified by the user which can override (Mask) a Match ID comparison at any particular bit (or, combination of bits) in an Acceptance Filter. Individual Masks, one for each Message Object, are programmed by the user in designated MMRs. Individual Mask patterns assure that single Receive Objects can Screen for multiple acknowledged CAL/CAN Frames and thus minimize the number of Receive Objects that must be dedicated to such lower priority Frames. This ability to Mask individual Message Objects is an important new CAL feature.

CAL: CAN Application Layer. A generic term for any high-level protocol which extends the capabilities of CAN while employing the CAN physical layer and the CAN frame format, and which adheres to the CAN specification. Among other things, CALs permit transmission of Messages which exceed the 8 byte data limit inherent to CAN Frames. This is accomplished by dividing each message into multiple packets, with each packet being transmitted as a single CAN Frame consisting of a maximum of 8 data bytes. Such messages are commonly referred to as "segmented" or "fragmented" messages. The individual CAN Frames constituting a complete fragmented message are not typically transmitted in a contiguous fashion, but rather, the individual CAN Frames of different, unrelated messages are interleaved on the CAN bus, as is illustrated in FIG. **2**

Fragmented Message: A lengthy message (in excess of 8 bytes) divided into data packets and transmitted using a sequence of individual CAN Frames. The specific ways that sequences of CAN Frames construct these lengthy messages is defined within the context of a specific CAL. The XA-C3 microcontroller automatically re-assembles these packets into the original, lengthy message in hardware and reports (via an interrupt) when the completed (re-assembled) message is available as an associated Receive Message Object.

Message Buffer: A block of locations in XA Data memory where incoming (received) messages are stored or where outgoing (transmit) messages are staged.

MMR: Memory Mapped Register. An on-chip command/control/status register whose address is mapped into XA Data memory space and is accessed as Data memory by the XA processor. With the XAC-3 microcontroller, a set of eight dedicated MMRs are associated with each Message Object. Additionally, there are several MMRs whose bits control global parameters that apply to all Message Objects.

With reference now to FIG. **3**, there can be seen a high-level block diagram of the XA-C3 microcontroller **20**. The XA-C3 microcontroller **20** includes the following functional blocks that are fabricated on a single integrated circuit (IC) chip packaged in a 44-pin PLCC or a 44-pin LQFP package:

- an XA CPU Core **22**, that is currently implemented as a 16-bit fully static CPU with 24-bit program and data address range, that is upwardly compatible with the 80C51architecture, and that has an operating frequency of up to 30 MHz;
- a program or code memory **24** that is currently implemented as a 32K ROM/EPROM, and that is bi-directionally coupled to the XA CPU Core **22** via an internal Program bus **25**. A map of the code memory space is depicted in FIG. **4**;
- a Data RAM **26** (internal or scratch pad data memory) that is currently implemented as a 1024 Byte portion of the overall XA-C3 data memory space, and that is bi-directionally coupled to the XA CPU Core **22** via an internal DATA bus **27**;
- an on-chip message buffer RAM or XRAM **28** that is currently implemented as a 512 Byte portion of the overall XA-C3 data memory space which may contain part or all of the CAN/CAL (Transmit & Receive Object) message buffers;
- a Memory Interface (MIF) unit **30** that provides interfaces to generic memory devices such as SRAM, DRAM, flash, ROM, and EPROM memory devices via an external address/data bus **32**, via an internal Core Data bus **34**, and via an internal MMR bus **36**;
- a DMA engine **38** that provides 32 CAL DMA Channels;
- a plurality of on-chip Memory Mapped Registers (MMRs) **40** that are mapped to the overall XA-C3 data memory space—a 4K Byte portion of the overall XA-C3 data memory space is reserved for MMRs. These MMRs include 32 (Message) Object or Address Pointers and 32 ID Screeners or Match IDs, corresponding to the 32 CAL Message Objects. A complete listing of all MMRs is provided in the Table depicted in FIG. **5**;

a 2.0B CAN/DLL Core **42** that is the CAN Controller Core from the Philips SJA1000CAN (2.0A/B) Data Link Layer (CDLL) device (hereinafter referred to as the "CAN Core Block" (CCB)); and,

an array of standard microcontroller peripherals that are bi-directionally coupled to the XA CPU Core **22** via a Special Function Register (SFR) bus **43**. These standard microcontroller peripherals include Universal Asynchronous Receiver Transmitter (UART) **49**, an SPI serial interface (port) **51**, three standard timers/counters with toggle output capability, namely, Timer **0** & Timer **1** included in Timer block **53**, and Timer **2** included in Timer block **54**, a Watchdog Timer **55**, and four 8-bit I/O ports, namely, Ports **0–3** included in block **61**, each of which has 4 programmable output configurations.

The DMA engine **38**, the MMRs **40**, and the CCB **42** can collectively be considered to constitute a CAN/CAL module **77**, and will be referred to as such at various times throughout the following description. Further, the particular logic elements within the CAN/CAL module **77** that perform "message management" and "message handling" functions will sometimes be referred to as the "message management engine" and the "message handler", respectively, at various times throughout the following description. Other nomenclature will be defined as it introduced throughout the following description.

As previously mentioned, the XA-C3 microcontroller **20** automatically implements, in hardware, many message management and other functions that were previously only implemented in software running on the host CPU (or not implemented at all), including transparent, automatic re-assembly of up to 32 concurrent, interleaved, multi-frame, fragmented CAL messages. For each application that is installed to run on the host CPU (i.e., the XA CPU Core **22**), the user (software programmer) must set-up the hardware for performing these functions by programming certain ones of the MMRs and SFRs in the manner set forth in the XA-C3 Functional Specification and XA-C3 CAN Transport Layer Controller User Manual. The register programming procedures that are most relevant to an understanding of the present invention are described below, followed by a description of the various message management and other functions that are automatically performed by the CAL/CAN module **77** during operation of the XA-C3 microcontroller **20** after it has been properly set-up by the user. Following these sections, a more detailed description of the particular invention to which this application is directed is provided.

Set-up/Programming Procedures

As an initial matter, the user must map the overall XA-C3 data memory space, as illustrated in FIG. **5**. In particular, subject to certain constraints, the user must specify the starting or base address of the XRAM **28** and the starting or base address of the MMRs **40**. The base address of the MMRs **40** can be specified by appropriately programming Special Function Registers (SFRs) MRBL and MRBH. The base address of the XRAM **28** can be specified by appropriately programming the MMRs designated MBXSR and XRAMB (see FIG. **4**).

The user can place the 4 KByte space reserved for MMRs **40** anywhere within the entire 16 Mbyte data memory space supported by the XA architecture, other than at the very bottom of the memory space (i.e., the first 1 KByte portion, starting address of 000000h), where it would conflict with the on-chip Data RAM **26** that serves as the internal or scratch-pad memory. The 4 KBytes of MMR space will always start at a 4K boundary. The reset values for MRBH and MRBL are 0Fh and F0h, respectively. Therefore, after a reset, the MMR space is mapped to the uppermost 4K Bytes of Data Segment 0Fh, but access to the MMRs **40** is disabled. The first 512 Bytes (offset 000h–1FFh) of MMR space are the Message Object Registers (eight per Message Object) for objects n=0–31, as is shown in FIG. **6**.

The base address of the XRAM **28** is determined by the contents of the MMRs designated MBXSR and XRAMB, as is shown in FIGS. **7** and **8**. As previously mentioned, the 512 Byte XRAM **28** is where some (or all) of the 32 (Rx/Tx) message buffers (corresponding to Message Objects n=0–31) reside. The message buffers can be extended off-chip to a maximum of 8 KBytes. This off-chip expansion capability can accommodate up to thirty-two, 256-Byte message buffers. Since the uppermost 8 bits of all message buffer addresses are formed by the contents of the MBXSR register, the XRAM **28** and all 32 message buffers must reside in the same 64K Byte data memory segment. Since the XA-C3 microcontroller **20** only provides address lines **A0–A19** for accessing external memory, all external memory addresses must be within the lowest 1 MByte of address space. Therefore, if there is external memory in the system into which any of the 32 message buffers will be mapped, then all 32 message buffers and the XRAM **28** must also be mapped entirely into that same 64K Byte segment, which must be below the 1 MByte address limit.

After the memory space has been mapped, the user can set-up or define up to 32 separate Message Objects, each of which can be either a Transmit (Tx) or a Receive (Rx) Message Object. A Rx Message Object can be associated either with a unique CAN ID, or with a set of CAN IDs which share certain ID bit fields. As previously mentioned, each Message Object has its own reserved block of data memory space (up to 256 Bytes), which is referred to as that Message Object's message buffer. As will be seen, both the size and the base address of each Message Object's message buffer is programmable.

As previously mentioned, each Message Object is associated with a set of eight MMRs **40** dedicated to that Message Object. Some of these registers function differently for Tx Message Objects than they do for Rx Message Objects. These eight MMRs **40** are designated "Message Object Registers" (see FIG. **4**).

The names of these eight MMRs **40** are:

1. MnMIDH Message n Match ID High
2. MnMIDL Message n Match ID Low
3. MnMSKH Message n Mask High
4. MnMSKL Message n Mask Low
5. MnCTL Message n Control
6. MnBLR Message n Buffer Location Register
7. MnBSZ Message n Buffer Size
8. MnFCR Message n Fragment Count Register

where n ranges from 0 to 31 (i.e., corresponding to 32 independent Message Objects).

In general, the user defines or sets up a Message Object by configuring (programming) some or all of the eight MMRs dedicated to that Message Object, as will be described below. Additionally, as will be described below, the user must configure (program) the global GCTL register, whose bits control global parameters that apply to all Message Objects.

In particular, the user can specify the Match ID value for each Message Object to be compared against the Screener IDs extracted from incoming CAN Frames for Acceptance Filtering. The Match ID value for each Message Object n is

specified in the MnMIDH and MnMIDL registers associated with that Message Object n. The user can mask any Screener ID bits which are not intended to be used in Acceptance Filtering, on an object-by-object basis, by writing a logic '1' in the desired (to-be-masked) bit position(s) in the appropriate MnMSKH and/or MNMSKL registers associated with each particular Message Object n. The user is responsible, on set-up, for assigning a unique message buffer location for each Message Object n. In particular, the user can specify the least significant 16 bits of the base address of the message buffer for each particular Message Object n by programming the MnBLR register associated with that Message Object n. The upper 8 bits of the 24-bit address, for all Message Objects, are specified by the contents of the MBXSR register, as previously discussed, so that the message buffers for all Message Objects reside within the same 64 KByte memory segment. The user is also responsible, on set-up, for specifying the size of the message buffer for each Message Object n. In particular, the user can specify the size of the message buffer for each particular Message Object n by programming the MnBSZ register associated with that Message Object n. The top location of the message buffer for each Message Object n is determined by the size of that message buffer as specified in the corresponding MnBSZ register.

The user can configure (program) the MnCTL register associated with each particular Message Object n in order to enable or disable that Message Object n, in order to define or designate that Message Object n as a Tx or Rx Message Object; in order to enable or disable automatic hardware assembly of fragmented Rx messages (i.e., automatic fragmented message handling) for that Message Object n; in order to enable or disable automatic generation of a Message-Complete Interrupt for that Message Object n; and, in order to enable or not enable that Message Object n for Remote Transmit Request (RTR) handling. In CANopen and OSEK systems, the user must also initialize the MnFCR register associated with each Message Object n.

As previously mentioned, on set-up, the user must configure (program) the global GCTL register, whose bits control global parameters that apply to all Message Objects. In particular, the user can configure (program) the GCTL register in order to specify the high-level CAL protocol (if any) being used (e.g., DeviceNet, CANopen, or OSEK); in order to enable or disable automatic acknowledgment of CANopen Frames (CANopen auto-acknowledge); and, in order to specify which of two transmit (Tx) pre-arbitration schemes/policies is to be utilized (i.e., either Tx pre-arbitration based on CAN ID, with the object number being used as a secondary tie-breaker, or Tx pre-arbitration based on object number only).

Receive Message Objects and the Receive Process

During reception (i.e., when an incoming CAN Frame is being received by the XA-C3 microcontroller **20**), the CAN/CAL module **77** will store the incoming CAN Frame in a temporary (13-Byte) buffer, and determine whether a complete, error-free CAN frame has been successfully received. If it is determined that a complete, error-free CAN Frame has been successfully received, then the CAN/CAL module **77** will initiate Acceptance Filtering in order to determine whether to accept and store that CAN Frame, or to ignore/discard that CAN Frame.

Acceptance Filtering

In general, because the XA-C3 microcontroller **20** provides the user with the ability to program separate Match ID and Mask fields for each of the 32 independent Message Objects, on an object-by-object basis, as described previously, the Acceptance Filtering process performed by the XA-C3 microcontroller **20** can be characterized as a "match and mask" technique. The basic objective of this Acceptance Filtering process is to determine whether a Screener ID field of the received CAN Frame (excluding the "don't care" bits masked by the Mask field for each Message Object) matches the Match ID of any enabled one of the 32 Message Objects that has been designated a Receive Message Object. If there is a match between the received CAN Frame and more than one Message Object, then the received CAN Frame will be deemed to have matched the Message Object with the lowest object number (n).

Acceptance Filtering is performed as follows by the XA-C3 microcontroller **20**:

(1) A Screener ID field is extracted from the incoming (received) CAN Frame. In this regard, the Screener ID field that is assembled from the incoming bit stream is different for Standard and Extended CAN Frames. In particular, as is illustrated in FIG. **9**, the Screener ID field for a Standard CAN Frame is 28 bits, consisting of 11 CAN ID bits extracted from the header of the received CAN Frame +2×8 (16) bits from the first and second data bytes (Data Byte **1** and Data Byte **2**) of the received CAN Frame +the IDE bit. Thus, the user is required to set the Msk**1** and Msk**0** bits in the Mask Field (MnMSKL register) for Standard CAN Frame Message Objects, i.e., to "don't care". In addition, in many applications based on Standard CAN Frames, either Data Byte **1**, Data Byte **2**, or both do not participate in Acceptance Filtering. In those applications, the user must also mask out the unused Data Byte(s). The IDE bit is not maskable. As is illustrated in FIG. **10**, the Screener ID field for an Extended CAN Frame is 30 bits, consisting of **29** CAN ID bits extracted from the header of the incoming CAN Frame +the IDE bit. Again, the IDE bit is not maskable.

(2) The assembled Screener ID field of the received CAN Frame is then sequentially compared to the corresponding Match ID values specified in the MnMIDH and MnMIDL registers for all currently enabled Receive Message Objects. Of course, any bits in the Screener ID field that are masked by a particular Message Object are not included in the comparison. That is, if there is a '1' in a bit position of the Mask field specified in the MnMSKH and Mn MSKL registers for a particular Message Object, then the corresponding bit position in the Match ID field for that particular Message Object becomes a "don't care", i.e., always yields a match with the corresponding bit of the Screener ID of the received CAN Frame.

(3) If the above comparison process yields a match with more than one Message Object, then the received CAN Frame will be deemed to have matched the Message Object having the lowest object number (n).

Message Storage:

Each incoming (received) CAN Frame that passes Acceptance Filtering, will be automatically stored, via the DMA engine **38**, into the message buffer for the Receive Message Object that particular CAN Frame was found to have matched. In an exemplary implementation, the message buffers for all Message Objects are contained in the XRAM **28**.

Message Assembly:

In general, the DMA engine **38** will transfer each accepted CAN Frame from the 13-byte pre-buffer to the appropriate message buffer (e.g., in the XRAM **28**), one word at a time, starting from the address pointed to by the contents of the MBXSR and MnBLR registers. Every time the DMA engine **38** transfers a byte or a word, it has to request the bus. In this

regard, the MIF unit **30** arbitrates between accesses from the XA CPU Core **22** and from the DMA engine **38**. In general, bus arbitration is done on an "alternate" policy. After a DMA bus access, the XA CPU Core **22** will be granted bus access, if requested. After an XA CPU bus access, the DMA engine **38** will be granted bus access, if requested. (However, a burst access by the XA CPU Core **22** cannot be interrupted by a DMA bus access).

Once bus access is granted by the MIF unit **30**, the DMA engine **38** will write data from the 13-byte pre-buffer to the appropriate message buffer location. The DMA engine **38** will keep requesting the bus, writing message data sequentially to the appropriate message buffer location until the whole accepted CAN Frame is transferred. After the DMA engine **38** has successfully transferred an accepted CAN Frame to the appropriate message buffer location, the contents of the message buffer will depend upon whether the message that the CAN Frame belongs to is a non-fragmented (single frame) message or a fragmented message. Each case is described below:

Non-Fragmented Message Assembly:

For Message Objects that have been set up with automatic fragmented message handling disabled (not enabled—i.e., the FRAG bit in the MnCTL register for that Message Object is set to '0'), the complete CAN ID of the accepted CAN Frame (which is either 11 or 29 bits, depending on whether the accepted CAN Frame is a Standard or Extended CAN Frame) is written into the MnMIDH and MnMIDL registers associated with the Message Object that has been deemed to constitute a match, once the DMA engine **38** has successfully transferred the accepted CAN Frame to the message buffer associated with that Message Object. This will permit the user application to see the exact CAN ID which resulted in the match, even if a portion of the CAN ID was masked for Acceptance Filtering. As a result of this mechanism, the contents of the MnMIDH and MNMIDL registers can change every time an incoming CAN Frame is accepted. Since the incoming CAN Frame must pass through the Acceptance Filter before it can be accepted, only the bits that are masked out will change. Therefore, the criteria for match and mask Acceptance Filtering will not change as a result of the contents of the MnMIDH and MnMIDL registers being changed in response to an accepted incoming CAN Frame being transferred to the appropriate message buffer.

Fragmented Message Assembly:

For Message Objects that have been set up with automatic fragmented message handling enabled (i.e., with the FRAG bit in the MnCTL register for that Message Object set to '1'), masking of the 11/29 bit CAN ID field is disallowed. As such, the CAN ID of the accepted CAN Frame is known unambiguously, and is contained in the MnMIDH and MnMIDL registers associated with the Message Object that has been deemed to constitute a match. Therefore, there is no need to write the CAN ID of the accepted CAN Frame into the MnMIDH and MnMIDL registers associated with the Message Object that has been deemed to constitute a match.

As subsequent CAN Frames of a fragmented message are received, the new data bytes are appended to the end of the previously received and stored data bytes. This process continues until a complete multi-frame message has been received and stored in the appropriate message buffer.

Under CAL protocols DeviceNet, CANopen, and OSEK, if a Message Object is an enabled Receive Message Object, and its associated MnCTL register has its FRAG bit set to '1' (i.e., automatic fragmented message assembly is enabled for that particular Receive Message Object), then the first data byte (Data Byte **1**) of each received CAN Frame that matches that particular Receive Message Object will be used to encode fragmentation information only, and thus, will not be stored in the message buffer for that particular Receive Message Object. Thus, message storage for such "FRAG-enabled" Receive Message Objects will start with the second data byte (Data Byte **2**) and proceed in the previously-described manner until a complete multiframe message has been received and stored in the appropriate message buffer. This message storage format is illustrated in FIG. **11**. The message handler hardware will use the fragmentation information contained in Data Byte **1** of each CAN Frame to facilitate this process.

Under the CAN protocol, if a Message Object is an enabled Receive Message Object, and its associated MnCTL register has its FRAG bit set to '1' (i.e., automatic fragmented message assembly is enabled for that particular Receive Message Object), then the CAN Frames that match that particular Receive Message Object will be stored sequentially in the message buffer for that particular Receive Message Object using the format shown in FIG. **12**.

When writing message data into a message buffer associated with a Message Object n, the DMA engine **38** will generate addresses automatically starting from the base address of that message buffer (as specified in the MnBLR register associated with that Message Object n). Since the size of that message buffer is specified in the MnBSZ register associated with that Message Object n, the DMA engine **38** can determined when it has reached the top location of that message buffer. If the DMA engine **38** determines that it has reached the top location of that message buffer, and that the message being written into that message buffer has not been completely transferred yet, the DMA engine **38** will wrap around by generating addresses starting from the base address of that message buffer again. Some time before this happens, a warning interrupt will be generated so that the user application can take the necessary action to prevent data loss.

The message handler will keep track of the current address location of the message buffer being written to by the DMA engine **38**, and the number of bytes of each CAL message as it is being assembled in the designated message buffer. After an "End of Message" for a CAL message is decoded, the message handler will finish moving the complete CAL message and the Byte Count into the designated message buffer via the DMA engine **38**, and then generate an interrupt to the XA CPU Core **22** indicating that a complete message has been received.

Since Data Byte **1** of each CAN Frame contains the fragmentation information, it will never be stored in the designated message buffer for that CAN Frame. Thus, up to seven data bytes of each CAN Frame will be stored. After the entire message has been stored, the designated message buffer will contain all of the actual informational data bytes received (exclusive of fragmentation information bytes) plus the Byte Count at location **00** which will contain the total number of informational data bytes stored.

It is noted that there are several specific user set-up/programming procedures that must be followed when invoking automatic hardware assembly of fragmented OSEK and CANopen messages. These and other particulars can be found in the XA-C3 CAN Transport Layer Controller User Manual that is part of the parent Provisional Application Serial No. 60/154,022, the disclosure of which has been fully incorporated herein for all purposes.

Transmit Message Objects and the Transmit Process

In order to transmit a message, the XA application program must first assemble the complete message and store it

in the designated message buffer for the appropriate Transmit Message Object n. The message header (CAN ID and Frame Information) must be written into the MnIMIDH, MNMIDL, and MnMSKH registers associated with that Transmit Message Object n. After these steps are completed, the XA application is ready to transmit the message. To initiate a transmission, the object enable bit (OBJ_EN bit) of the MnCTL register associated with that Transmit Message Object n must be set, except when transmitting an Auto-Acknowledge Frame in CANopen. This will allow this ready-to-transmit message to participate in the pre-arbitration process. In this connection, if more than one message is ready to be transmitted (i.e., if more than one Transmit Message Object is enabled), a Tx Pre-Arbitration process will be performed to determine which enabled Transmit Message Object will be selected for transmission. There are two Tx Pre-Arbitration policies which the user can choose between by setting or clearing the Pre_Arb bit in the GCTL register.

After a Tx Message Complete interrupt is generated in response to a determination being made by the message handler that a completed message has been successfully transmitted, the Tx Pre-Arbitration process is "reset", and begins again. Also, if the "winning" Transmit Message Object subsequently loses arbitration on the CAN bus, the Tx Pre-Arbitration process gets reset and begins again. If there is only one Transmit Message Object whose OBJ_EN bit is set, it will be selected regardless of the Tx Pre-Arbitration policy selected.

Once an enabled Transmit Message Object has been selected for transmission, the DMA engine **38** will begin retrieving the transmit message data from the message buffer associated with that Transmit Message Object, and will begin transferring the retrieved transmit message data to the CCB **42** for transmission. The same DMA engine and address pointer logic is used for message retrieval of transmit messages as is used for message storage of receive messages, as described previously. Further, message buffer location and size information is specified in the same way, as described previously. In short, when a transmit message is retrieved, it will be written by the DMA engine **38** to the CCB **42** sequentially. During this process, the DMA engine **38** will keep requesting the bus; when bus access is granted, the DMA engine **38** will sequentially read the transmit message data from the location in the message buffer currently pointed to by the address pointer logic; and, the DMA engine **38** will sequentially write the retrieved transmit message data to the CCB **42**. It is noted that when preparing a message for transmission, the user application must not include the CAN ID and Frame Information fields in the transmit message data written into the designated message buffer, since the Transmit (Tx) logic will retrieve this information directly from the appropriate MnMIDH, MnMIDL, and MnMSKH registers.

The XA-C3 microcontroller **20** does not handle the transmission of fragmented messages in hardware. It is the user's responsibility to write each CAN Frame of a fragmented message to the appropriate message buffer, enable the associated Transmit Message Object for transmission, and wait for a completion before writing the next CAN Frame of that fragmented message to the appropriate message buffer. The user application must therefore transmit multiple CAN Frames one at a time until the whole multi-frame, fragmented transmit message is successfully transmitted. However, by using multiple Transmit Message Objects whose object numbers increase sequentially, and whose CAN IDs have been configured identically, several CAN Frames of a fragmented transmit message can be queued up and enabled, and then transmitted in order.

To avoid data corruption when transmitting messages, there are three possible approaches:

1. If the Tx Message Complete interrupt is enabled for the transmit message, the user application would write the next transmit message to the designated transmit message buffer upon receipt of the Tx Message Complete interrupt. Once the interrupt flag is set, it is known for certain that the pending transmit message has already been transmitted.
2. Wait until the OBJ_EN bit of the MnCTL register of the associated Transmit Message Object clears before writing to the associated transmit message buffer. This can be accomplished by polling the OBJ_EN bit of the MnCTL register of the associated Transmit Message Object.
3. Clear the OBJ_EN bit of the MnCTL register of the associated Transmit Message Object while that Transmit Message Object is still in Tx Pre-Arbitration.

In the first two cases above, the pending transmit message will be transmitted completely before the next transmit message gets transmitted. For the third case above, the transmit message will not be transmitted. Instead, a transmit message with new content will enter Tx Pre-Arbitration.

There is an additional mechanism that prevents corruption of a message that is being transmitted. In particular, if a transmission is ongoing for a Transmit Message Object, the user will be prevented from clearing the OBJ_EN bit in the MnCTL register associated with that particular Transmit Message Object.

CAN/CAL RELATED INTERRUPTS

The CAN/CAL module **77** of the XA-C3 microcontroller **20** is presently configured to generate the following five different Event interrupts to the XA CPU Core **22**:

1. Rx Message Complete
2. Tx Message Complete
3. Rx Buffer Full
4. Message Error
5. Frame Error

For single-frame messages, the "Message Complete" condition occurs at the end of the single frame. For multi-frame (fragmented) messages, the "Message Complete" condition occurs after the last frame is received and stored. Since the XA-C3 microcontroller **20** hardware does not recognize or handle fragmentation for transmit messages, the Tx Message Complete condition will always be generated at the end of each successfully transmitted frame.

As previously mentioned, there is a control bit associated with each Message Object indicating whether a Message Complete condition should generate an interrupt, or just set a "Message Complete Status Flag" (for polling) without generating an interrupt. This is the INT_EN bit in the MnCTL register associated with each Message Object n.

There are two 16-bit MMRs **40**, MCPLH and MCPLL, which contain the Message Complete Status Flags for all 32 Message Objects. When a Message Complete (Tx or Rx) condition is detected for a particular Message Object, the corresponding bit in the MCPLH or MCPLL register will be set. This will occur regardless of whether the INT_EN bit is set for that particular Message Object (in its associated MnCTL register), or whether Message Complete Status Flags have already been set for any other Message Objects.

In addition to these 32 Message Complete Status Flags, there is a Tx Message Complete Interrupt Flag and an Rx Message Complete Interrupt Flag, corresponding to bits [1]

and [0], respectively, of an MMR **40** designated CANINTFLG, which will generate the actual Event interrupt requests to the XA CPU Core **22**. When an End-of-Message condition occurs, at the same moment that the Message Complete Status Flag is set, the appropriate Tx or Rx Message Complete Interrupt flip-flop will be set provided that INT_EN=1 for the associated Message Object, and provided that the interrupt is not already set and pending.

Further details regarding the generation of interrupts and the associated registers can be found in the XA-C3 Functional Specification and in the XA-C3 CAN Transport Layer Controller User Manual, both of which are part of the parent Provisional Application Serial No. 60/154,022, the disclosure of which has been fully incorporated herein for all purposes.

THE PRESENT INVENTION

With all presently available CAN devices, the entire device, including the processor core, the CAN/CAL module, and other peripheral components and blocks, must all be awake (i.e., powered-on and active), whenever any CAN activity is detected or in progress, e.g., whenever a CAN message is being received, transmitted, assembled, handled, stored, filtered, or otherwise processed, because CAL software running on the host CPU must actively participate in these functions, e.g., the CAL software running on the host CPU actively monitors and manages the buffering and processing of the message data, and the assembly of multi-frame, fragmented messages. In this regard, with all existing CAN devices, any power conservation mode of operation, such as idle, sleep, or power-down mode of operation, must be invoked on a system-wide level, in order for the CAN device to perform any CAN-related activity, such as assembling multi-frame, fragmented messages.

As will be apparent from the above and foregoing description, the XA-C3 microcontroller of the preferred embodiment, in addition to providing a number of enhanced features and additional capabilities with respect to CAL/CAN message management and handling, primarily performs these CAL/CAN message management and handling functions, including automatic assembly of multi-frame, fragmented messages, in hardware, thereby reducing the CPU CAL/CAN message processing overhead ("CAL instruction bandwidth") from approximately 80% (with the presently available technology) to as low as 10% (with the XA-C3 microcontroller **20**). Because of this novel architecture, the XA CPU Core **22** (sometimes referred to hereinafter simply as the "processor core") of the XA-C3 microcontroller **20**, and various other peripherals and sub-blocks of the CCB **42**, are not required to be awake during significant periods of time while certain of these functions are being performed.

In accordance with the present invention, the XA-C3 microcontroller **20** supports a reduced power mode of operation known as Idle mode, which significantly reduces power consumption. In this regard, just putting the processor core **22**, by itself, into Idle mode reduces power consumption by approximately 15 mA at 30 MHz.

In overview, in the Idle mode, the processor core **22** is "halted" (put to "sleep") and clocks to the processor core **22** are stopped to conserve power. The term "stopped" as used here means that the clocks are disabled, shut down, or gated-off to the block or component being put to sleep, e.g., the processor core **22**. Clocks to some peripheral blocks are stopped as well, while other peripherals (e.g., timers) continue to operate. The Idle mode is terminated upon occurrence of any interrupt or in the event of a system reset.

In the XA-C3 microcontroller **20**, the CAN/CAL module **77** constitutes a large proportion of the overall chip area and, hence, is responsible for a commensurately large proportion of the power consumption. Given this, it is highly desirable to shut down this module during Idle mode if it is not actually in use. If the CAL/CAN module **77** is in use, however, it may not be shut off until it has completed whatever tasks it may be handling. Moreover, if any new activity is detected on the CAN bus after the CAN/CAL module **77** is shut down, it is essential that the CAN/CAL module **77** immediately wake up to handle the incoming message. It was recognized that the CAN/CAL module **77** could be taken into and out of its own "Idle mode" at will, without any need to restart the processor core **22**. The processor core **22** (and other select peripherals) can remain in power-saving ("Idle") mode indefinitely while the CAN/CAL module **77** can be brought out of the "Idle mode" on an as-needed basis and returned to "sleep" whenever its functionality is not immediately required.

In accordance with the present invention, all of the logic (hereinafter referred to as the "sleep control module") required to implement the "Idle" mode is included in the CCB **42**. In the present implementation, the default condition for the CAN/CAL module **77** will be to stay awake in Idle mode, so that the processor core **22** can "sleep" while CAN transmissions or receptions, or associated message management activities, are in progress. Any interrupt, e.g., an Rx Message Complete interrupt, a Tx Message Complete interrupt, an Rx Buffer Full interrupt, a Message Error interrupt, a Frame Error interrupt, or any other internal or external interrupt, will wake up the processor core **22**. An option is provided to enable the user to include the CAN/CAL module **77** in the Idle mode. This option ("CAL/CAN module sleep enable") can be selected by the user in software by setting the SLPEN bit [3] in the MMR **40** designated CANCMR (CAN Command Register). With this option invoked, i.e., with the CAL/CAN module sleep enable function activated, whenever the XA-C3 microcontroller **20** is in the Idle mode, the CAL sleep control module **91** (see FIG. **13**) continuously polls all of the sub-components or sub-blocks comprising the CAN/CAL module **77** to determine if any of them are currently engaged in message-handling activity. If no ongoing activity is detected, or as soon as any activity ends, the sleep control module **91** will stop the clock to the entire CAN/CAL module **77**, thereby putting it to "sleep". Subsequently, if a signal transition is detected on the CAN Rx pin **93**, the clocks to the CAN/CAL module **77** will be instantly re-enabled and full operation of this module will be restored so that it can begin receiving the incoming frame. However, no interrupt will be generated, and the processor core **22** will remain asleep.

Once the entire CAN frame is received and (if necessary) stored, the CAN/CAL module **77** will typically go back to sleep until a new message frame is detected on the CAN bus (i.e., until a signal transition is detected on the CAN Rx pin **93**). When the XA-C3 microcontroller **20** is in the Idle mode, the processor core **22** and the rest of the XA-C3 microcontroller **20** will only be awoken in response to a normal interrupt once a complete message has been received and assembled (unless, of course, some other system interrupt wakes it up prior to that time). With the present implementation of the XA-C3 microcontroller **20**, wake-up from Idle mode is instantaneous, and is initiated via any interrupt; and, I_{dd} in the Idle mode is in the range of 20–30

mA if the CAN/CAL module **77** is deactivated (put to sleep), and approximately 60–120 mA if the CAN/CAL module **77** is left active (awake).

A key feature of the XA-C3 CAN/CAL module is its unique ability to automatically assemble in hardware long ("fragmented") CAL messages which are transmitted over many individual CAN frames. Given this, it is possible, and even likely, that the processor core **22** can remain in its power saving mode (i.e., asleep during the "Idle" mode) for a very long period of time after CAN bus activity has started before it is needed to process a completed message. During this time, the CAN/CAL module **77** may go into and out of its own power-saving mode (i.e., sleep state) repeatedly. In this regard, the processor core **22** and the CAN/CAL module **77** can be considered to each have an Idle mode, with the processor core **22** remaining in its Idle mode while the CAN/CAL module **77** is repeatedly brought into and out of its Idle mode. The only conditions that must be met for the CAN/CAL module **77** to be safely put to sleep (Idle or Power-Down mode) is that there be no CAN activity in progress and no interrupts pending (i.e., the processor core **22** must itself already be in its Idle mode). "Power-Down mode" in the present implementation of the XA-C3 microcontroller **20** means that the main oscillator (not shown) is clamped-off and there is no chip activity of any kind. I_{dd} in this mode is on the order to a few tens of microamps. Wake-up from the Power-Down mode is accomplished via a system reset or a transition on the External Interrupt 0 or 1 pins (not shown). The wake-up period is 10,000 oscillator clocks, which is enough time for several CAN frames to be transmitted. If a transition of the CAN RxD input occurs when the XA-C3 microcontroller **20** is in the Power-Down mode, the processor core **22** will enter Idle mode (after a **9892** clock delay), and the CAN/CAL module **77** will be activated to receive and process the incoming frame. When the CAN/CAL module **77** generates an interrupt (or some other enabled interrupt occurs), only then will the processor core **22** come out of its Idle mode and begin executing code. Code execution will resume either in the interrupt service routine, if its priority is higher than current code, or with the next instruction following the Power-Down instruction. At this time, the termination of the Power-Down mode is actually complete.

As previously mentioned, the term "sleep control module" refers to the logic circuitry contained within the CCB **42** that is required to implement the "Idle" mode. With reference now to FIG. **13**, the sleep control module **91** will now be described. As can be seen, the sleep control module **91** includes AND gates **95** and **97**, and an asynchronous latch **99**. The asynchronous latch **99** has an input that is coupled to the CAN Rx pin **93**. The asynchronous latch **99** is presently implemented as cross-coupled NOR gates (not shown). The output ("No Rx") of this latch **99** is normally at a logic high ('1') level. When any signal transition occurs on the CAN Rx pin **93**, this signal transition clears the asynchronous latch **99**, to thereby drive its output No Rx to a logic low ('0') level.

In operation, each individual sub-block a (to be described later) within the CAN/CAL module **77** supplies a "sleepok$_n$" signal back to the sleep control module **91** to assert its readiness to be shut down (i.e., to indicate that it is not engaged in any CAN message-handling activity). In particular, assuming there are n sub-blocks , then n sleepok$_n$ signals are produced. Each of these sleepok$_n$ signals indicates that the providing sub-block is currently inactive and, insofar as that particular sub-block is concerned, it is okay to put the CAN/CAL module **77** to sleep. These n sleepok$_n$ signals are applied as respective inputs to the AND gate **95**. The resulting "sleep_enable" signal generated at the output of the AND gate **95** will therefore be active (logic '1') only if all of the component sub-blocks assert their individual "sleepok$_n$" signals, i.e., only if all of the sleepok$_n$ signals are at a logic high ('1') level, indicating that all of the individual sub-blocks within the CAN/CAL module **77** are currently inactive and ready to be put to sleep. The individual sub-blocks within the CAN/CAL module **77** that generate a sleepok$_n$ signal in the present implementation of the XA-C3 microcontroller **20** are depicted in FIG. **14**. These sub-blocks include the DMA engine **38**, a Tx Pre-Arbitration sub-block **101**, a Message Management sub-block **103**, a Tx Logic sub-block **105**, and a Message Pointer/Handler sub-block **107**.

The "sleep_enable" signal is then logically AND'ed with a global "idle_mode" signal provided by the processor core **22** when it has no pending interrupts. In particular, the sleep_enable and idle_mode signals are applied as first and second inputs to the AND gate **97**. The third input to this AND gate **97** is the output No Rx of the asynchronous latch **99**. As previously mentioned, the output No Rx of the asynchronous latch **99** will be active (logic high) by default, and will only go inactive (logic low) in response to a signal transition on the CAN Rx pin **93**, i.e., when an incoming message is being received.

Thus, the output "ccb_idle_n" signal generated by the AND gate **97** will be active (logic high) only if all three of its inputs, i.e., the sleep_enable, idle_mode, and No Rx signals, are active (logic high). In order for this to occur, therefore, three conditions must be met:

1. The processor core **22** itself must already be in its Idle mode;
2. All of the component sub-blocks within the CAN/CAL module **77** must assert their willingness to be shut down; and,
3. No incoming message is being received.

The output of this AND gate **97** serves as a clock disable signal "ccb_idle_n", which directly shuts off the clock to the entire CAN/CAL module **77** when it is at a logic high ('1') level. The clearing of the asynchronous latch **99** in response to detection of an incoming message will instantly de-activate the clock disable signal ccb_idle_n, resulting in the immediate activation of the clocks to the CAN/CAL module **77**. At this point, the CAN/CAL module **77** will be fully functional and can begin receiving this incoming message. As soon as the last bit of the incoming frame is received, the asynchronous latch **99** will be set back to a logic '1' so that it will no longer interfere, i.e., disable the clock, since its output "No Rx" will also be active (logic '1'). By this time, however, one of more sub-blocks within the CAN/CAL module **77** will be actively engaged in handling the incoming frame and will, accordingly, have lowered its "sleepok$_n$" line. Once handling of the frame is completed, assuming the frame was not the final frame of an enabled message and assuming no further activity on the CAN bus, the clocks to the CAN/CAL module **77** will again be disabled and will return to its Idle mode. When the final frame of a message is ultimately received and stored, the CAN/CAL module **77** will generate a standard "message-complete" interrupt request to the processor core **22**. As with any other interrupt, this request will terminate the global idle_mode condition (i.e., drive the idle_mode signal low), and wake up the processor core **22** and any sleeping sub-blocks in the CAN/CAL module **77**.

In addition, the MMR **40** designated CANSTR (Can Status Register) includes a bit, CAL_SLEEP_OK, that is

readable by the processor core **22**, and that, when set ('1') indicates that it is OK to put the CAN/CAL module **77** to sleep. In this connection, if the processor core **22** is about to put the XA-C3 microcontroller **20** into Power-Down mode, it must first read this bit (CAL_SLEEP_OK) from the CANSTR register to determine if it is safe to do so. There is no need for the processor core **22** to read this bit prior to entering its Idle mode, as the processor core **22** is free to go into its Idle mode whenever it chooses, independently of the CAN/CAL module **77**. The CAN/CAL module **77** will follow if and when it is ready.

Although the present invention has been described in detail hereinabove in the context of a specific preferred embodiment/implementation, it should be clearly understood that many variations, modifications, and/or alternative embodiments/implementations of the basic inventive concepts taught herein which may appear to those skilled in the pertinent art will still fall within the spirit and scope of the present invention, as defined in the appended claims.

What is claimed is:

1. A method for conserving power in a CAN microcontroller, formed by a single integrated circuit (IC), that includes a processor core and a CAN/CAL module that includes a plurality of sub-blocks that cooperatively function to process incoming CAL/CAN messages, the method comprising the steps of:

placing the processor core in a power-reduction mode of operation;

placing the CAN/CAL module in a power-reduction mode of operation; and,

activating the CAN/CAL module to process an incoming CAL/CAN message, thereby terminating the power-reduction mode of operation thereof, while the processor core is in its power-reduction mode of operations;

wherein the CAN/CAL module is placed in the power-reduction mode of operation only after each respective sub-block of the plurality of sub-blocks of the CAN/CAL module indicates a current inactive state, said processor core indicates a current inactive state, and no incoming CAL/CAN message is detected.

2. The method as set forth in claim **1**, wherein the incoming CAL/CAN message is a multi-frame, fragmented CAL/CAN message.

3. The method as set forth in claim **2**, further comprising the step of the CAN/CAL module performing automatic hardware assembly of the multi-frame, fragmented CAL/CAN message while the processor core is its power-reduction mode of operation.

4. The method as set forth in claim **3**, further comprising the steps of:

generating a message-complete interrupt in response to completion of assembly of the multi-frame, fragmented CAL/CAN message; and

activating the processor core in response to the message-complete interrupt.

5. The method as set forth in claim **1**, further comprising repeating the step of placing the CAN/CAL module in a power-reduction mode of operation and activating the CAN/CAL module to process an incoming CAL/CAN message, in seriatim, a plurality of times, while the processor core is in its power reduction mode of operation.

6. The method as set forth in claim **1**, wherein the power-reduction mode of operation comprises a sleep state.

7. The method as set forth in claim **1**, wherein the power reduction mode of operation comprises an idle mode of operation.

8. The method as set forth in claim **1**, wherein the step of placing the processor core in a power reduction mode of operation comprises:

generating a clock disable signal having a first logic level if the processor core has pending interrupts, and having a second logic if the processor core has no pending interrupts; and

disabling a clock applied to the processor core in response to the clock disable signal to thereby place the processor core in the power-reduction mode of operation.

9. A method for conserving power in a CAN microcontroller that includes a processor core and a CAN/CAL module that includes a plurality of sub-blocks that cooperatively function to process incoming CAL/CAN messages, the method comprising the steps of:

placing the processor core in a power-reduction mode of operation;

placing the CAN/CAL module in a power-reduction mode of operation; and,

activating the CAN/CAL module to process an incoming CAL/CAN message, thereby terminating the power-reduction mode of operation thereof, while the processor core is in its power-reduction mode of operation;

wherein the step of placing the CAN/CAL module in a power-reduction mode operation comprises:

each of the plurality of sub-blocks generating a respective first signal having a first logic level if that sub-block is currently active, and having a second logic level if that sub-block us not currently active;

generating a second signal having a first logic level if any of the first signals are at the first logic level, and having a second logic level in response to all of the first signals having the second logic level;

generating a third signal having a first logic level if the processor core is not idle, and having a second logic level if the processor core is idle;

generating a fourth signal having a first logic level if an incoming message is being received, and having a second logic level if an incoming message is not being received;

generating a clock disable signal in response to the second, third, and fourth signals all being at their respective second logic level; and

disabling a clock applied to the CAN/CAL module in response to the clock disable signal to thereby place the CAN/CAL module in the power-reduction mode of operation.

10. A method for conserving power in a CAN microcontroller that includes a processor core and a CAN/CAL module that includes a plurality of sub-blocks that cooperatively function to process incoming CAL/CAN messages, the method comprising the steps of:

placing the processor core in a power-reduction mode of operation;

placing the CAN/CAL module in a power-reduction mode of operation; and,

activating the CAN/CAL module to process an incoming CAL/CAN message, thereby terminating the power-reduction mode of operation thereof, while the processor core is in its power-reduction mode of operation;

wherein the step of placing the CAN/CAL module in a power reduction mode operation comprises:

each of the plurality of sub-blocks generating a respective first signal having a first logic level if that sub-block is currently active, and having a second logic level if that sub-block is not currently active;

generating a second signal having a first logic level if any of the first signals are at the first logic level, and having a second logic level in response to all of the first signals having the second logic level;

generating a third signal having a first logic level if the processor core has pending interrupts, and having a second logic level if the processor core has no pending interrupts;

generating a fourth signal having a first logic level if an incoming message is being received, and having a second logic level if an incoming message is not being received;

generating a clock disable signal in response to the second, third, and fourth signals all being at their respective second logic level; and

disabling a clock applied to the CAN/CAL module in response to the clock disable signal to thereby place the CAN/CAL module in the power-reduction mode of operation.

11. The method as set forth in claim **10**, further comprising the step of activating the processor core in response to an interrupt.

12. A method for conserving power in a CAN microcontroller, formed by a single integrated circuit (IC), that includes a processor core and a CAN/CAL module, the method comprising the steps of:

placing the processor core in a power-reduction mode of operation while the CAN/CAL module is actively processing an incoming CAL/CAN message; and,

terminating the power-reduction mode of operation in response to an interrupt;

wherein the CAN/CAL module automatically assembles incoming, multi-frame, fragmented messages while the processor core remains in its power-reduction mode of operation; and further comprising the step of:

placing the CAN/CAL module in a power-reduction mode of operation only after confirming that each respective sub-block of a plurality of sub-blocks of the CAN/CAL module indicates a current inactive state, said processor core indicates a current inactive state, and no incoming CAL/CAN message is detected.

13. The method as set forth in claim **12**, wherein the power-reduction mode of operation comprises a sleep state.

14. The method as set forth in claim **12**, wherein the power-reduction mode of operation comprises an idle mode of operation.

15. The method as set forth in claim **12**, wherein the CAN/CAL module generates a message-complete interrupt in response to completion of assembly of the multi-frame, fragmented message, whereby the terminating step is executed in response to the message-complete interrupt.

16. The method as set forth in claim **12**, wherein the step of placing the processor core in a power-reduction mode of operation comprises:

generating a clock disable signal having a first logic level if the processor core has pending interrupts, and having a second logic level if the processor core has no pending interrupts; and

disabling a clock applied to the processor core in response to the clock disable signal to thereby place the processor core in the power-reduction mode of operation.

17. A method for conserving power in a CAN microcontroller, formed by a single integrated circuit (IC), that includes a processor core and a CAN/CAL module, the method comprising the steps of:

placing the entire CAN microcontroller, including both the processor core and the CAN/CAL module in a power-down mode of operation only after detecting that each respective sub-block of the plurality of sub-blocks of the CAN/CAL module indicates a current inactive state, said processor core indicates a current inactive state, and no incoming CAL/CAN message is detected;

detecting receipt of an incoming message; and,

activating the CAN/CAL module in response to the detecting step to process the incoming message, thereby terminating the power-down mode of operation of the CAN/CAL module, without terminating the power-down mode of operation of the processor core.

18. The method as set forth in claim **17**, further comprising the step of placing the processor core in a power reduction mode of operation in response to the detecting step, wherein:

the processor core can be substantially instantaneously woken up when in the power-down mode of operation; and

the processor core can be woken up over a prescribed wake-up period when in the power-down mode of operation.

19. The method as set forth in claim **17**, wherein the placing step comprises:

determining whether the CAN/CAL module is ready to be placed into the power-down mode of operation; and

stopping a main system clock in response to a determination that the CAN/CAL module is ready to be placed into the power-down mode of operation.

20. The method as set forth in claim **18**, wherein the CAN/CL module automatically assembles incoming, multi-frame, fragmented messages while the processor core remains in its power-reduction mode of operation.

21. The method as set forth in claim **17**, further comprising the step of terminating the power-down mode of operation of the entire CAN microcontroller, including both the processor core and the CAN/CAL module, in response to an external interrupt.

22. The method as set forth in claim **18**, further comprising the step of terminating the power-down mode of operation of the entire CAN microcontroller, including both the processor core and the CAN/CAL module, in response to system reset command.

23. A CAN microcontroller, formed by a single integrated circuit (IC), comprising:

a processor core that runs CAN applications;

a CAN/CAL module that includes a plurality of sub-blocks that cooperatively function to process incoming CAL/CAN messages, the CAN/CAL module including a power control module that includes:

means for placing the processor core in a power-reduction mode of operation;

means for placing the CAN/CAL module in a power-reduction mode of operation; and,

means for activating the CAN/CAL module to process an incoming CAL/CAN message, thereby terminating the power-reduction mode of operation thereof, while the processor core is in its power-reduction mode of operation; wherein

the CAN/CAL module is placed in the power-reduction mode of operation only after each respective sub-block of the plurality of sub-blocks of the CAN/CAL module indicates a current inactive state, said processor core

indicates a current inactive state, and no incoming CAL/CAN message is detected.

24. The CAN microcontroller as set forth in claim **23**, wherein the incoming CAL/CAN message is a multi-frame, fragmented CAL/CAN message.

25. The CAN microcontroller as set forth in claim **24**, wherein the CAN/CAL module is operable to perform automatic hardware assembly of the multi-frame, fragmented CAL/CAN message while the processor core is in its power-reduction mode of operation.

26. The CAN microcontroller as set forth in claim **25**, wherein the power control module further includes:

means for generating a message-complete interrupt in response to completion of assembly of the multi-frame, fragmented CAL/CAN message; and

means for activating the processor core in response to the message-complete interrupt.

27. The CAN microcontroller as set forth in claim **23**, wherein the power-reduction mode of operation comprises a sleep state.

28. The CAN microcontroller as set forth in claim **23**, wherein the power-reduction mode of operation comprises an idle mode of operation.

29. The CAN microcontroller as set forth in claim **23**, wherein the means for placing the CAN/CAL module in a power-reduction mode operation comprises:

means for generating a clock disable signal having a first logic level if the processor core has pending interrupts, and having a second logic level if the processor core has no pending interrupts; and

means for disabling a clock applied to the processor core in response to the clock disable signal to thereby place the processor core in the power-reduction mode of operation.

30. A CAN microcontroller, formed by a single integrated circuit (IC), comprising:

a processor core that runs CAN applications;

a CAN/CAL module that includes a plurality of sub-blocks that cooperatively function to process incoming CAL/CAN messages, the CAN/CAL module including a power control module that includes:

means for placing the processor core in a power-reduction mode of operation;

means for placing the CAN/CAL module in a power-reduction mode of operation; and,

means for activating the CAN/CAL module to process an incoming CAL/CAN message, thereby terminating the power-reduction mode of operation thereof, while the processor core is in its power-reduction mode of operation;

wherein the means for placing the CAN/CAL module in a power-reduction mode operation comprises:

means associated with each of the plurality of sub-blocks for generating a respective first signal having a first logic level if that sub-block is currently active, and having a second logic level if that sub-block is not currently active;

means for generating a second signal having a first logic level if any of the first signals are at the first logic level, and having a second logic level in response to all of the first signals having the second logic level;

means for generating a third signal having a first logic level if the processor core is not idle, and having a second logic level if the processor core is idle;

means for generating a fourth signal having a first logic level if an incoming message is being received, and having a second logic level if an incoming message is not being received;

means for generating a clock disable signal in response to the second, third, and fourth signals all being at their respective second logic level; and

means for disabling a clock applied to the CAN/CAL module in response to the clock disable signal to thereby place the CAN/CAL module in the power-reduction mode of operation.

31. A CAN microcontroller, formed by a single integrated circuit (IC) comprising:

a processor core that runs CAN applications;

a CAN/CAL module that includes a plurality of sub-blocks that cooperatively function to process incoming CAL/CAN messages, the CAN/CAL module including a power control module that includes:

means for placing the processor core in a power-reduction mode of operation;

means for placing the CAN/CAL module in a power-reduction mode of operation; and,

means for activating the CAN/CAL module to process an incoming CAL/CAN message, thereby terminating the power-reduction mode of operation thereof, while the processor core is in its power-reduction mode of operation;

wherein the means for placing the CAN/CAL module in a power-reduction mode operation comprises:

means associated with each of the plurality of sub-blocks for generating a respective first signal having a first logic level if that sub-block is currently active, and having a second logic level if that sub-block is not currently active;

means for generating a second signal having a first logic level if any of the first signals are at the first logic level, and having a second logic level in response to all of the first signals having the second logic level;

means for generating a third signal having a first logic level if the processor core has pending interrupts, and having a second logic level if the processor core has no pending interrupts;

means for generating a fourth signal having a first logic level if an incoming message is being received, and having a second logic level if an incoming message is not being received;

means for generating a clock disable signal in response to the second, third and fourth signals all being at their respective second logic level; and

means for disabling a clock applied to the CAN/CAL module in response to the clock disable signal to thereby place the CAN/CAL module in the power-reduction mode of operation.

32. A CAN microcontroller, formed by a single integrated circuit (IC), comprising:

a processor core that runs CAN applications;

a CAN/CAL module that processes incoming CAL/CAN messages, the CAN/CAL module including a power control module that includes:

means for placing the processor core in a power-reduction mode of operation while the CAN/CAL module is actively processing an incoming CAL/CAN message; and,

means for terminating the power-reduction mode of operation of said processor core in response to an interrupt; and

means for placing the CAN/CAL module in a power-reduction mode, each respective sub-block of a plural-

ity of sub-blocks of the CAN/CAL module indicates a current inactive state, said processor core indicates a current inactive state, and no incoming CAL/CAN message is detected.

33. The CAN microcontroller as set forth in claim **32**, wherein the CAN/CAL module automatically assembles incoming, multi-frame, fragmented messages while the processor core remains in its power-reduction mode of operation.

34. The CAN microcontroller as set forth in claim **33**, wherein the CAN/CAL module generates the interrupt in response to completion of assembly of the multi-frame, fragmented CAL/CAN message.

35. The CAN microcontroller as set forth in claim **32**, wherein the means for placing the processor core in a power-reduction mode of operation comprises:

means for generating a clock disable signal having a first logic level if the processor core has pending interrupts, and having a second logic level if the processor core has no pending interrupts; and

means for disabling a clock applied to the processor core in response to the clock disable signal to thereby place the processor core in the power reduction mode of operation.

36. A CAN microcontroller, formed by a single integrated circuit (IC), comprising:

a processor core that runs CAN applications;

a CAN/CAL module that processes incoming CAL/CAN messages, the CAN/CAL module including a power control module that includes:

means for placing the entire CAN microcontroller, including both the processor core and the CAN/CAL module in a power-down mode of operation;

means for detecting receipt of an incoming message; and,

means for activating the CAN/CAL module in response to detection of the receipt of an incoming message, to process the incoming message, thereby terminating the power-down mode of operation of the CAN/CAL module, without terminating the power-down mode of operation of the processor core;

wherein the means for placing CAN microcontroller in a power-down mode of operation includes means for determining that each respective sub-block of a plurality of sub-blocks of the CAN/CAL module indicates a current inactive state, said processor core indicates a current inactive state, and no incoming CAL/CAN message is detected before the CAN/CAL module is placed in the power-reduction mode of operation.

37. The CAN microcontroller as set forth in claim **36**, wherein the power control module further includes means for placing the processor core in a power-reduction mode of operation in response to detection of the receipt of an incoming message, wherein:

the processor core can be substantially instantaneously woken up when in the power-down mode of operation; and

the processor core can be woken up over a prescribed wake-up period when in the power-down mode of operation.

38. The CAN microcontroller as set forth in claim **36**, wherein the means for placing comprises:

means for determining whether the CAN/CAL module is ready to be placed into the power-down mode of operation; and

means for stopping a main system clock in response to a determination that the CAN/CAL module is ready to be placed into the power-down mode of operation.

39. The CAN microcontroller as set forth in claim **37**, wherein the CAN/CAL module automatically assembles incoming, multi-frame, fragmented messages while the processor core remains in its power-reduction mode of operation.

40. The CAN microcontroller as set forth in claim **36**, wherein the power control module further includes means for terminating the power-down mode of operation of the entire CAN microcontroller, including both the processor core and the CAN/CAL module, in response to an external interrupt.

41. The CAN microcontroller as set forth in claim **36**, wherein the power control module further includes means for terminating the power-down mode of operation of the entire CAN microcontroller, including both the processor core and the CAN/CAL module, in response to a system reset command.

42. A CAN microcontroller, formed by a single integrated circuit (IC), comprising:

a processor core that runs CAN applications;

a CAN/CAL module that includes a plurality of sub-blocks that cooperatively function to process incoming CAL/CAN messages, the CAN/CAL module including a power control module that includes:

first logic circuitry that places the processor core in a power-reduction mode of operation;

second logic circuitry that places the CAN/CAL module in a power-reduction mode of operation; and,

third logic circuitry that activates the CAN/CAL module to process an incoming CAL/CAN message, thereby terminating the power-reduction mode of operation thereof, while the processor core is in its power-reduction mode of operation;

wherein the second logic circuitry places the CAN/CAL module in a power reduction mode of operation only after determining each respective sub-block of a plurality of sub-blocks of the CAN/CAL module indicates a current inactive state, said processor core indicates a current inactive state, and no incoming CAL/CAN message is detected.

43. A CAN microcontroller, formed by a single integrated circuit (IC), comprising:

a processor core that runs CAN applications;

a CAN/CAL module that processes incoming CAL/CAN messages, the CAN/CAL module including a power control module that includes:

first logic circuitry that places the processor core in a power-reduction mode of operation while the CAN/CAL module is actively processing an incoming CAL/CAN message, wherein the first logic circuitry places the CAN/CAL module in a power reduction mode of operation only after determining that each respective sub-block of a plurality of sub-blocks of the CAN/CAL module indicates a current inactive state, said processor core indicates a current inactive state, and no incoming CAL/CAN message is detected subsequent to the active processing of a current CAL/CAN message; and,

second logic circuitry that terminates the power-reduction mode of operation in response to an interrupt.

44. A CAN microcontroller, formed by a single integrated circuit (IC), comprising:

a processor core that runs CAN applications;

a CAN/CAL module that processes incoming CAL/CAN messages, the CAN/CAL module including a power control module that includes:

first logic circuitry that places the entire CAN microcontroller, including both the processor core and the CAN/CAL module in a power-down mode of operation;

second logic circuitry that detects receipt of an incoming message; and,

third logic circuitry that activates the CAN/CAL module in response to detection of the receipt of an incoming message, to process the incoming message, thereby terminating the power-down mode of operation of the CAN/CAL module, without terminating the power-down mode of operation of the processor core;

wherein the first logic circuitry places the CAN/CAL module in a power reduction mode of operation only after determining that each respective sub-block of a plurality of sub-blocks of the CAN/CAL module indicates a current inactive state, said processor core indicates a current inactive state, and no incoming CAL/CAN message is detected.

* * * * *